Ingeniería de software 1

Autor: Fredy Alonso León Socha

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Ingeniería de software 1 / Fredy Alonso León Socha, / Bogotá D.C.,
Fundación Universitaria del Área Andina. 2017

978-958-5459-11-3

Catalogación en la fuente Fundación Universitaria del Área Andina (Bogotá).

Edición:
Fondo editorial Areandino
Fundación Universitaria del Área Andina
Calle 71 11-14, Bogotá D.C., Colombia
Tel.: (57-1) 7 42 19 64 ext. 1228
E-mail: publicaciones@areandina.edu.co
http://www.areandina.edu.co

Primera edición: noviembre de 2017

Corrección de estilo, diagramación y edición: Dirección Nacional de Operaciones virtuales
Diseño y compilación electrónica: Dirección Nacional de Investigación

Hecho en Colombia
Made in Colombia

Ingeniería de software 1

Autor: Fredy Alonso León Socha

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

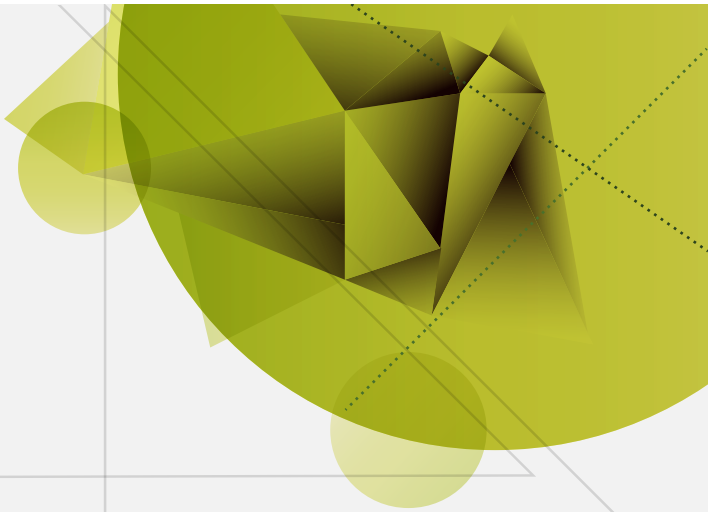
Índice

Índice

1

Unidad 1

Introducción a la Ingeniería de software



Ingeniería de software

Autor: Fredy Alonso León Socha

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

El presente módulo tiene como objetivo dar a conocer como ha sido el proceso evolutivo de la información, desde sus comienzos; en el paleolítico; hasta cuando aparecieron las primeras computadoras.

Se inicia con las primeras formas utilizadas para comunicación, como los pictogramas y se continúa con el proceso evolutivo de la información antes, durante y después de la revolución industrial. Finalmente se presentan los eventos más importantes en la evolución de las telecomunicaciones.

Estas temáticas le ayudaran a saber de dónde han surgido las tecnologías que conocemos actualmente, pues la mayoría se basan o han sido una evolución de tecnologías que se han venido utilizado o se han generado hace mucho tiempo.

U1

Metodología

El módulo se basa en la lectura individual de la presente cartilla y posterior desarrollo de actividades planteadas en la plataforma online.

Se espera una participación y acceso a plataforma a su consideración, como mejor pueda organizarse, para cubrir los objetivos de aprendizaje y participación. Por mi experiencia, recomiendo la asistencia diaria para aprovechar al máximo el curso. Es interesante seguir los hilos de participación 2-3 veces al día y dedicar uno momento para analizar y participar inteligentemente.

La comunicación conmigo la pueden realizar abiertamente a través del Foro y personalmente a través del email.

U1 Desarrollo temático

Introducción a la Ingeniería de software

Definiciones

Conjunto de conocimientos y técnicas científicas, empíricas y prácticas aplicadas a la invención, el diseño, el desarrollo, la construcción, el mantenimiento y el perfeccionamiento de tecnologías, estructuras, máquinas, herramientas, sistemas, materiales y procesos para la resolución de problemas prácticos.

Software

Conjunto de programas, instrucciones y reglas informáticas que permiten que se puedan ejecutar distintos procesos o tareas específicas en un computador.

Programador/Desarrollador

Persona encargada de escribir el código necesario en un lenguaje de programación que pueda ser entendido por una computadora, usando para ello algoritmos y estructuras de datos.

Ingeniero de software

Persona utilizan técnicas de ingeniería para el diseño, especificación, codificación, validación y administración de tiempos y presupuestos establecidos en un proyecto de desarrollo.

Analista de negocio

Se encarga de entender lo que quiere el cliente y lo transforma en requerimientos.

Arquitecto de software

Ve el software a nivel de diseño y transforma los requerimientos.

Analista de calidad

Verifica que el software entregado por el desarrollador, funcione bien y cumpla los requerimientos del cliente.

Jefe de proyecto

Gestiona el equipo de desarrollo.

Evolución del software

A continuación se muestra los eventos que han marcado un referente en la evolución cronológica del software. Este ha estado muy ligado al proceso evolutivo del hardware computacional, pues en la medida que se generaban componentes más pequeños y más rápidos, el software necesariamente debía ajustarse a dichos recursos físicos, a fin de obtener el mayor rendimiento.

1950 a 1965

- Los procesos de desarrollo de software los realizaban las mismas organizaciones para sus propios requerimientos. No existía documentación de soporte, no había una planificación y los programadores tenían que trabajar con los pocos métodos que existían.

1965 a 1975

- Los procesos de desarrollo de software los realizaban las mismas organizaciones para sus propios requerimientos. No existía documentación de soporte, no había una planificación y los programadores tenían que trabajar con los pocos métodos que existían.

1975 a 1985

- Se empezaron a realizar procesos de planificación para el desarrollo.
- La ejecución de funciones permitían la comunicación de información entre múltiples computadoras, lo que se le llamo Procesamiento Distribuido.
- Los requerimientos empezaron a centrarse en acceder a los datos en forma instantánea.
- El software empezó a aplicarse en "sistemas inteligentes" , los cuales podían realizar procesos sin intervención humana, por ej. los hornos microondas, robots, etc.

1985 a 2000

- Se masifico la programación orientada a objetos.
- Se comenzó a utilizar el software para simular el procesamiento de información basada en redes neuronales artificiales.
- Se crearon los llamados entornos cliente-servidor.
- Se empezaron a implementar mejores prácticas y etapas de desarrollo de software.
- La información se centralizaba en internet y de allí se recogían o transportaban datos desde y hacia diferentes sistemas ,por lo que las aplicaciones web comenzaron a tener un mayor auge.

Importancia del software

Para abordar este tema, es necesario abordarlo desde diferentes puntos de vista, de acuerdo al uso que se le dé.

En educación	Permite iniciar procesos de autoformación. Permite el acceso a la información.
En actividades diarias	En este caso, estaría enfocado a las capacidades que permiten los dispositivos móviles.
En el hogar	Optimizar tareas. Sin el software, una lavadora no podría llevar a cabo correctamente la labor para la cual fue diseñada y fabricada.
En el ámbito empresarial	Permite mejorar proceso. Mejorar la eficiencia. Incrementar el desarrollo y los alcances de la empresa. Permite la automatización de procesos. Optimizar los tiempos.

Si se habla de software libre, se podría decir que es importante porque:

Genera Autonomía tecnológica, ya que los usuarios pasan de tener un rol meramente de consumo a ser creadores de nuevos.

Permite la estandarización e integración: al ser producido por estándares y especificaciones libres y públicos, permite que fácilmente pueda ser ajustado a los requerimientos de una entidad o proceso determinado.

Permite una mayor seguridad: al manejar información abierto se conoce que información maneja y como lo hace, a quien la comparte, etc. Por otro lado, los hacker no ven como objetivo generar virus pues son fácilmente detectables por la misma naturaleza del código abierto.

Genera Independencia de proveedores: ya no se necesitará del fabricante para obtener actualizaciones, pues es el mismo usuario es quien las realiza.

Disminuye costos: el costo de obtener software libre es mucho menor que el coste de obtener una licencia comercial. De esta manera se optimizan los recursos económicos ya sea para usuarios o para empresas.

Problemas del software

Abarcaremos esta temática sin plantear problemas específicos de una determinada solución de software, si no que se realizara una generalización de problemas comunes en el desarrollo de software, sea cual sea su objetivo.

Falta de planificación

Una mala planificación implica que se extiendan los tiempos de entrega del producto, se incrementen los costos de personal al no tener bien definido el perfil del personal necesario, diseñador, grafico, de multimedia, fotógrafo, etc. Al final los pocos integrantes del equipo resultan haciendo funciones que nada tienen que ver con su rol o perfil profesional.

La estimación de costos

Nunca se sabe cuánto tiempo y trabajo implicara realmente un desarrollo hasta que no se empiecen a realizar y es en este proceso donde muchas veces no se tuvieron en cuenta los costos de horas de desarrollo que eran necesarias para lograr el objetivo.

Calidad en el producto final

Cuando se inicia con las pruebas finales, empiezan a salir a flote deficiencias en el mismo, pues este normalmente ha sido probado en condiciones optimas y por personal capacitado, pero la realidad es que el usuario o cliente final, es quien realmente debe hacer las pruebas finales, en el entorno en el que finalmente va a funcionar.

Estos problemas generan insatisfacción, falta de confianza en la empresa o equipo de desarrollo contratado, etc. Un cliente solo mira el resultado.

Altos costos de mantenimiento

Puede que un producto de software cumpla los requerimientos solicitados por un cliente, pero seguramente con el tiempo requerirá de ajustes, mejoras en funcionalidades, etc. y esto genera nuevos costos de desarrollo, independientemente si es para actualización o ajustes.

Requerimientos no claros

Normalmente el desarrollo de software se inicia con una indicación superficial de los requisitos por parte del cliente, sin haber indagado a profundidad, cada uno de ellos, sus implicaciones. En la marcha, surgen las dudas respecto a cómo debe desarrollarse, o como debería, generando retrasos, pérdidas de tiempo, tanto del equipo de desarrollo como del cliente.

El cliente normalmente una idea abstracta del resultado final, pero no sobre las funciones que debería cumplir el software.

A continuación se presenta una muy buena imagen que contextualiza la falta de comunicación en un desarrollo de software.

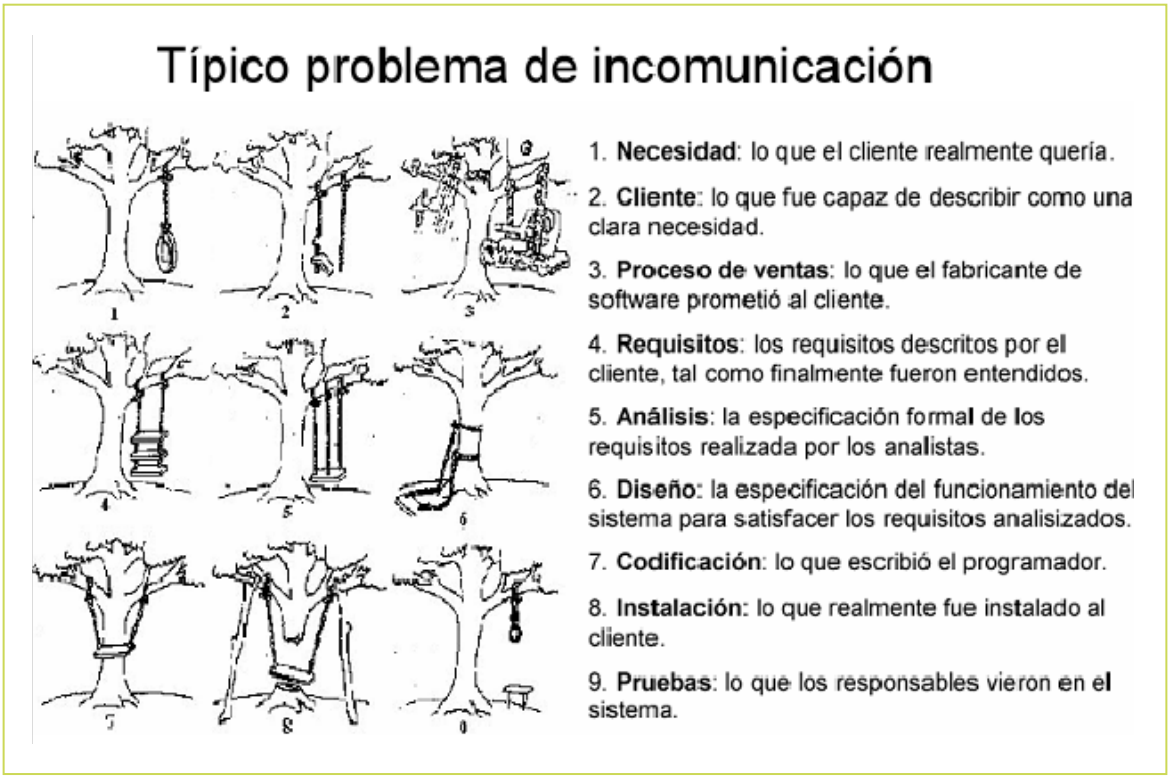


Imagen 1
Fuente: http://image.slidesharecdn.com/01-elprocesodedesarrollodesoftware-120911090848-phpapp01/95/01-el-procesodedesarrollodesoftware-5-728.jpg?cb=1347354573

Falta de documentación

¿Que pasaría si el programador, se va de la empresa o deja el proyecto? Si no hay documentación al respecto, seguramente quien tome las riendas tendrá que invertir tiempo en «entender» como se hizo, generando pérdidas de tiempo y dinero para la empresa.

Por otro lado si un cliente no tiene documentación de cómo funciona, seguramente se generaran problemas de uso y abuso del software, salvo en aquellos casos en el mismo, sea altamente intuitivo.

Características del software

Se definen las siguientes categorías: operativas, de transición, de revisión.

Características operativas

Se refieren a la forma como es presentado el software al cliente final o usuario y se incluyen los siguientes aspectos:

Corrección: realizar los ajustes necesarios a fin que cumpla con todos los requerimientos que el cliente ha establecido.

Usabilidad: se refiere a que sea sencillo de utilizar o sea fácil de aprender a usarlo.

Integridad: una vez instalado, no debe afectar el normal funcionamiento de otros programas o tareas del equipo donde se encuentre instalado.

Fiabilidad: el producto de software no debería tener ningún defecto. No sólo esto, no debe fallar mientras la ejecución.

Eficiencia: debe optimizar los recursos disponibles en el ambiente en que se ejecuta. Por ejemplo, usar eficazmente el espacio de almacenamiento, comunicación con otros sistemas, etc.

Seguridad: debe contar un control de la información almacenada, de tal forma que esté protegido contra ataques externos.

Características de transición

Interoperabilidad: permitir el intercambio de información con otros sistemas o aplicaciones, de tal forma que permita la escalabilidad.

Reutilización: el código debe estar escrito de tal forma que pueda reutilizarse en otros desarrollos, realizando pequeñas modificaciones o ajustes.

Portabilidad: que pueda ejecutar en diferentes plataformas o entornos, las funciones incluidas en el desarrollo.

Características de revisión

Se refieren a factores internos de funcionamiento como su estructura, eficiencia y documentación. Desglosando algunos aspectos, se pueden relacionar:

Mantenimiento: debe poder realizarse sin complicaciones

Flexibilidad: capacidad para que los ajustes o modificaciones se puedan realizar sin problemas.

Extensibilidad: aumentar su escalabilidad, debe ser fácil de incluir nuevas funciones.

Escalabilidad: debe ser muy fácil de actualizar para más trabajo.

Capacidad de prueba: pruebas del software debe ser fáciles.

Modularidad: debe estar compuesto por unidades y módulos independientes entre sí.

Conceptos de calidad

Características para medir la calidad del software

Utilidad

Que cumpla los requerimientos exigidos por el cliente o usuario.

Confiabilidad

Que permita realizar las funciones preestablecidas, en las condiciones reales de uso, durante un tiempo determinado, sin que se afecte la funcionalidad y manteniendo la integralidad de datos que se manejen.

Claridad

Deben ser fáciles de entender: internamente y externamente.

Económico

Que su desarrollo, uso y mantenimiento se pueda costear. El software debe optimizar y disminuir los recursos humanos, de tiempo o materiales que se requerían antes de haber sido adquirido.

Gestión de la calidad del software

La gestión de calidad del software implica dos procesos fundamentales: el control de calidad y el aseguramiento de la calidad.

Control de calidad del software

Son las técnicas y actividades operativas que se utilizan para realizar una verificación de requisitos que permitan mantener un control del proceso de desarrollo y eliminar todas aquellas causas que puedan generar algún defecto o anomalía, en alguna de las fases del ciclo del vida del producto de software.

Aseguramiento de la calidad

Implica todas las actividades que se requieren para que el producto de software pueda generar confianza y cumpla a cabalidad con los requerimientos planteados.

Dentro de estas actividades se pueden mencionar:

Revisiones técnicas y de gestión, a fin de evaluar (su objetivo es la evaluación).

Inspección, con el fin de verificar si se está construyendo el producto que ha solicitado el cliente.

Pruebas y validación, para determinar si el software se está construyendo en forma correcta.

Auditorias, para confirmar el cumplimiento de funciones.

A continuación se muestra un grafico donde se resumen los procesos incluidos dentro de la Gestión de calidad. Cada una de las etapas implica un desarrollo conjunto de estrategias o actividades que permiten lograr la calidad de software y todas.

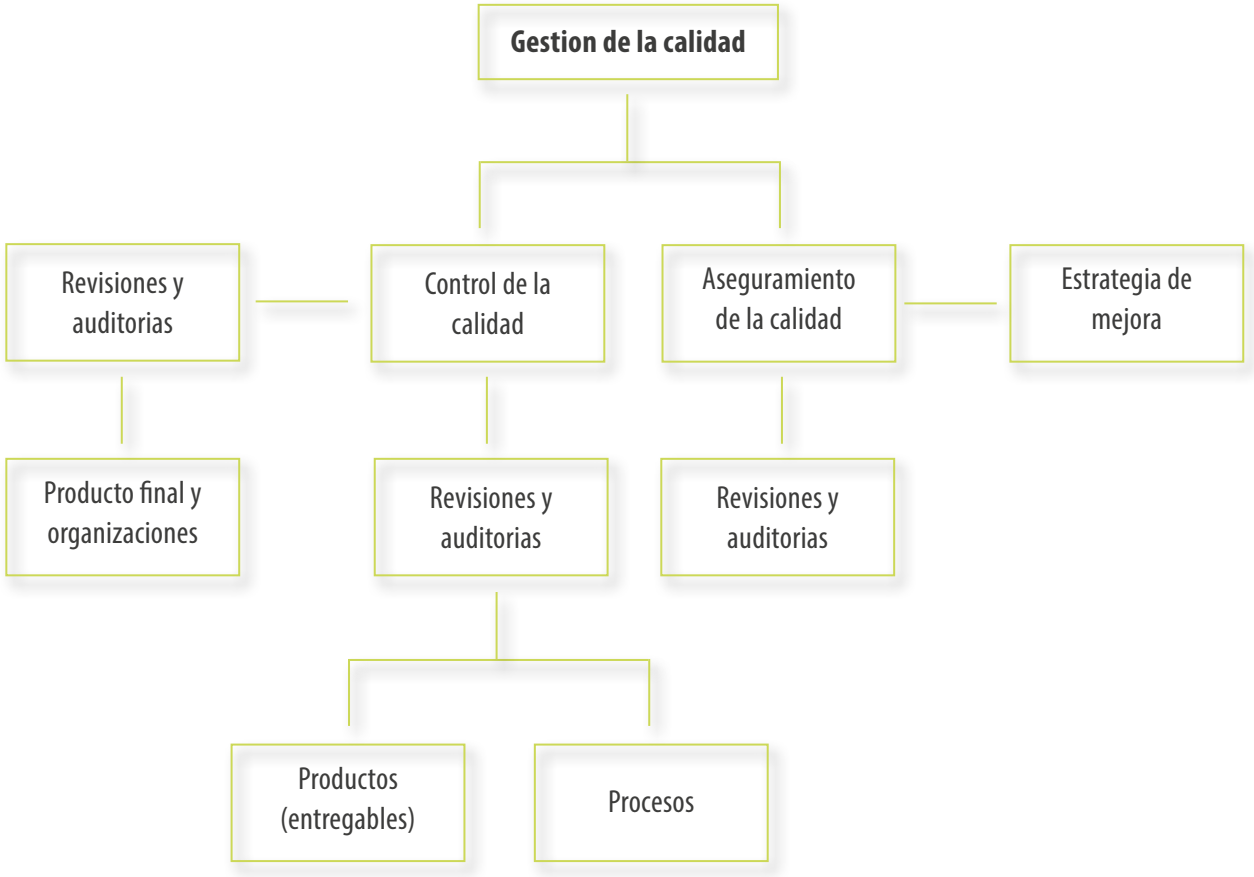
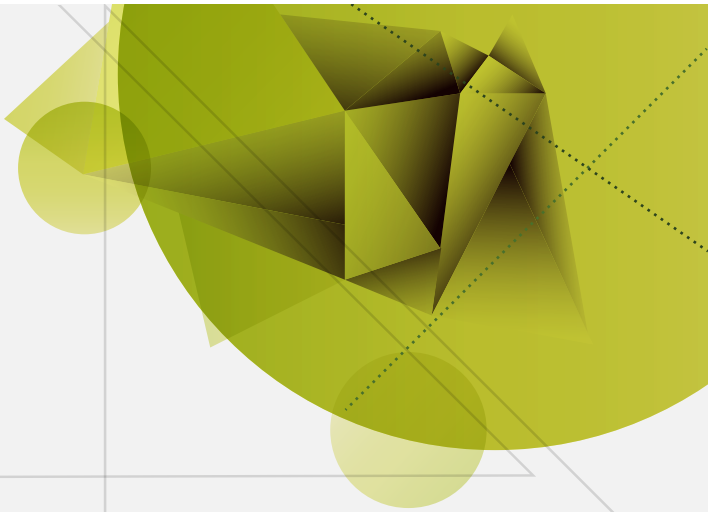


Figura 1
Fuente: Propia.

1

Unidad 1

Introducción a la Ingeniería de software



Ingeniería de software

Autor: Fredy Alonso León Socha

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

La presente cartilla muestra la evolución que han tenido las telecomunicaciones, los medios de transmisión usados en la actualidad y una introducción a los conceptos básicos de electricidad y electrónica. Una situación practica de las temáticas expuestas, es a la hora de implementación de un sistema de información, ya que usted deberá conocer cuál es el consumo de corriente o voltaje de los equipos para en base a esto, utilizar un cable de conexión especifico que soporte esa carga y poder realizar la implementación de la red.

Por otro lado, al conocer las características ventajas, o desventajas de un medio de transmisión u otro, podrá determinar cuál es la mejor alternativa a la hora de la implementación de un sistema de comunicación.

U1

Metodología

El módulo se basa en la lectura individual de la presente cartilla y posterior desarrollo de actividades planteadas en la plataforma online.

Se espera una participación y acceso a plataforma a su consideración, como mejor pueda organizarse, para cubrir los objetivos de aprendizaje y participación. Por mi experiencia, recomiendo la asistencia diaria para aprovechar al máximo el curso. Es importante seguir los hilos de participación 2-3 veces al día y dedicar uno momento para analizar y participar inteligentemente.

La comunicación conmigo la pueden realizar abiertamente a través del Foro y personalmente a través del email.

Introducción a la Ingeniería de software

Mitos del software

Mitos del cliente

Mitos del cliente	
Mito	Realidad
Sé que me hago entender con comentarles la idea de lo que necesito.	Se requiere de una excelente comunicación entre el cliente y analista ,de tal forma que pueda establecer claramente cada uno de los requerimientos que el cliente necesita para su software: información, interfaces gráficas, funciones, rendimiento, diseño, ejercicios de validación, etc.
Agregar una nueva funcionalidad es sencillo, pues me dijeron que el software es flexible.	Si los cambios son solicitados cuando el software ya está en proceso de diseño, pueden incrementarse significativamente los costos de desarrollo (tiempos de entrega, horas de desarrollo, mas personal técnico, etc.), pues en esta etapa ya están definidos los recursos que se necesitan y también se tiene un diseño inicial sobre el que se está trabajando. Si el cliente solicita cambios al final del proyecto, los costos serán mucho más altos.

Cuadro 1
Fuente: Propia.

Mitos del desarrollador

Mitos del desarrollador	
Mito	Realidad
Para que metodologías, si lo que se necesita es que el software funcione y listo...ahí terminamos.	Mas del 50% del esfuerzo que se le dedica a un desarrollo, se realiza, después que al cliente se le ha entregado el primer prototipo, si se ha seguido alguna metodología tradicional como: RUP (Rational Unified Procces), MSF (Microsoft Solution Framework), Win-Win Spiral Model, Iconix. Por lo que para disminuir ese porcentaje de esfuerzo, es necesario implementar metodologías agiles de desarrollo, por ejemplo: XP (Extreme Programming), Scrum, Crystal Clear, DSDM (Dynamic Systems Developmemt Method), FDD (Feature Driven Development), ASD (Adaptive Software Development), XBreed, Extreme Modeling.
Solo podemos comprobar la calidad hasta que tengamos el software funcionando.	Las actividades que permiten una revisión técnica del software es un filtro de calidad que se debe implementar durante todo el proyecto, ya que permite detectar defectos de funcionamiento.
Solo necesitamos entregarle al cliente su software funcionando y ya!	El software implica el programa como tal, los datos que manejan el programa y la documentación del mismo. La documentación permite contar con una guía para las labores de mantenimiento, haciendo que estas se puedan hacer en un tiempo más corto.

Cuadro 2
Fuente: Propia.

Distribución del esfuerzo en un proyecto de programación

Grafica de distribución del esfuerzo

Se entiende por esfuerzo a la cantidad de recursos humanos que se requieren para llevar a cabo las tareas de un proyecto de software, donde ya se ha realizado un análisis documentado y detallado de todos los requerimientos solicitados por el cliente. Este esfuerzo es medido en meses/hombre.

Un proyecto de software involucra las siguientes etapas:

- Análisis y diseño: Se proyectan interfaces gráficas, casos de uso, funcionalidades, etc.
- Codificación: escribir el código fuente que permita llevar a cabo lo que se determino en la etapa de análisis y diseño.
- Pruebas: verificar si lo que se codifico y ejecuta el software cumple con los requerimientos.
- Adaptación: reutilizar código.
- Mejoras: incluir nuevas funcionalidades.
- Arreglos: corrección de errores.

Cada una de estas etapas se ha llevado a la siguiente una gráfica que representa el porcentaje de esfuerzo de cada una, durante el ciclo de desarrollo. Los porcentajes.

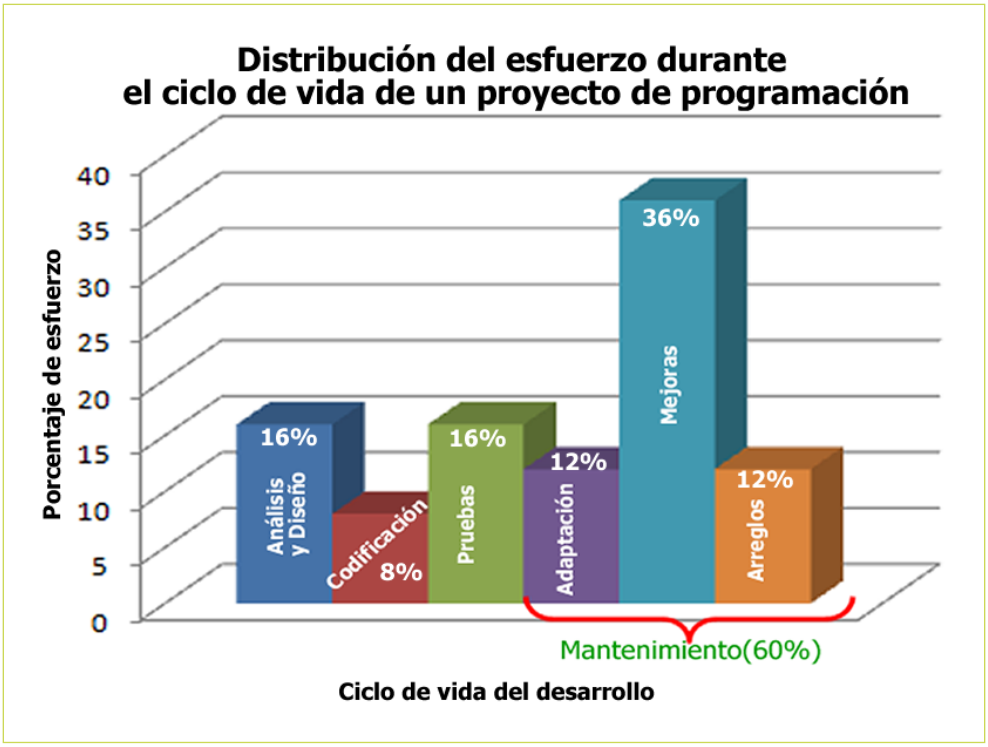


Imagen 1
Fuente: Propia.

Teniendo en cuenta la grafica anterior, se puede determinar que un 60% del esfuerzo está concentrado en las etapas de adaptación, mejoras y arreglos, es decir a mejorar el producto.

Opciones actuales para estimación

Usar proyectos similares

Puede funcionar, siempre y cuando los proyectos de referencia tengan gran similitud con el que se está realizando. El problema es que se puede llegar a necesitar datos estadísticos que no siempre están disponibles o que aunque estén disponibles, no representan indicadores exactos de los que requiere el proyecto.

Utilizar técnicas de descomposición sencillas

Descomponer el proyecto en secciones más pequeñas, de acuerdo a la descomposición del problema y descomposición del proceso (Pressman, 2005). Es importante que se conozca profundamente el alcance del software que se pretende construir, para luego realizar un proceso de estimación del tamaño del mismo.

Desarrollar un modelo empírico

Se basa en utilizar formulas empíricas para proyectar costos, esfuerzos y tiempos de desarrollo. Los datos han sido obtenidos de muestras limitadas de otros proyectos.

Usar herramientas automáticas

Permiten estimar costos y esfuerzos, duración, riesgos, análisis de situaciones de acuerdo a fechas de entrega, selección del personal, etc.

Los modelos más comunes de estimación

COCOMO II

Constructive Cost Model II (segunda versión). Considera la construcción de prototipos, el desarrollo basado en componentes y el uso de programación con bases de datos, para el desarrollo de software. Se compone de tres sub modelos:

- Composición de aplicaciones. Se utilizan componentes reutilizables, scripts y programación de bases de datos, para realizar estimaciones de desarrollo de prototipos.
- Diseño Inicial. Se utiliza en las primeras etapas de diseño, basándose en la exploración de diferentes arquitecturas del sistema y conceptos de operación, siempre y cuando los requerimientos ya hayan sido definidos.
- Post – Arquitectura. Se utiliza cuando ya se ha finalizado el diseño del sistema y se tiene su respectiva documentación detallada. Permite realizar estimaciones para las diferentes etapas del ciclo de vida de desarrollo.

Ecuación del software de Putnam

Modelo multivariable dinámico que fue planteado por Putnam (Pressman, 2005) y asume una distribución especifica del esfuerzo durante el ciclo de vida del mismo. Maneja la siguiente ecuación:

$$E = [\, LDC \times B^{0,333} / P \,]^3 \times (1/t^4)$$

Donde;

L= Esfuerzo en personas mes o personas / año.

t= Duración del proyecto en meses o años.

B= Factor especial de destrezas.

P= Parámetro de productividad.

Según [Putnam, 1992], en la medida que aumentan las necesidades de integración, garantías de calidad, pruebas, documentación y habilidad de gestión se aumenta el factor B.

El parámetro P refleja el estado de madurez de:

- Proceso de desarrollo.
- Prácticas de gestión.
- Uso correcto de las normas de la ingeniería de software.
- El nivel de los lenguajes de programación utilizados.
- Habilidades y experiencia del equipo.
- Complejidad de la aplicación.

Administración de proyectos de software

Conceptos básicos

Administrar un proyecto de software consiste en llevar un control del desarrollo, plazos y recursos establecidos. Involucra también la organización técnica y dirección del recurso humano que conforma el equipo de trabajo.

La administración de un proyecto de software involucra 4 áreas:

- Estructura. Con cuales elementos se conformará la organización.
- Proceso administrativo. Asignación de responsabilidades y actividades de supervisión del personal.
- Proceso de desarrollo. Los métodos, lenguajes de programación, herramientas a utilizar, fuentes de documentación y apoyo.
- Programación. Cronograma para realizar las actividades propuestas.

Y debe controlar los siguientes factores:

- El costo total del proyecto: aumentar/disminuir gastos o costos.
- Capacidades del proyecto: agregar/eliminar funcionalidades.
- Calidad del producto: como aumentar el tiempo entre fallos de una cierta severidad.
- Duración del proyecto: aumentar/reducir tiempo programado de terminación.

Secuencia de actividades fundamentales de administración

- Comprender el contenido, alcance y tiempos del proyecto: para tener un entendimiento global ya que lo especifico corresponde al personal técnico
- Identificar el proceso de desarrollo: métodos, herramientas, lenguajes, documentación, ayudas.
- Determinar la estructura organizativa: elementos de la organización involucrados y su interacción. Ej.: departamentos, compañías, lideres
- Identificar el proceso administrativo: como se llevará a cabo el modelo de organización y las responsabilidades que cada participante tiene dentro del proyecto.
- Programar el proceso: cronograma para realizar las actividades propuestas, con tiempos establecidos para cada una.
- Establecer equipo de trabajo: debe estar conformado de acuerdo a las actividades a ejecutar.
- Analizar los riesgos y buscar soluciones: esto ayuda a garantizar que el proyecto tenga éxito.
- Identificación de productos finales: debe establecerse los productos de documentación o código que deben generarse, los cuales deben estar asociados al cronograma de actividades

Tipos de estructura organizativa en equipos de trabajo

Organización jerarquía

Figura 1
Fuente: Propia

Consta de un director general del proyecto y bajo su mando se encuentran 3 líneas: Experto en mercado, Equipo de desarrollo y un área de calidad. Tiene como ventaja que las líneas de autoridad y decisión están bien definidas, el inconveniente es que los miembros ubicados en la parte inferior, tienen una baja participación en las decisiones.

Organización homogénea

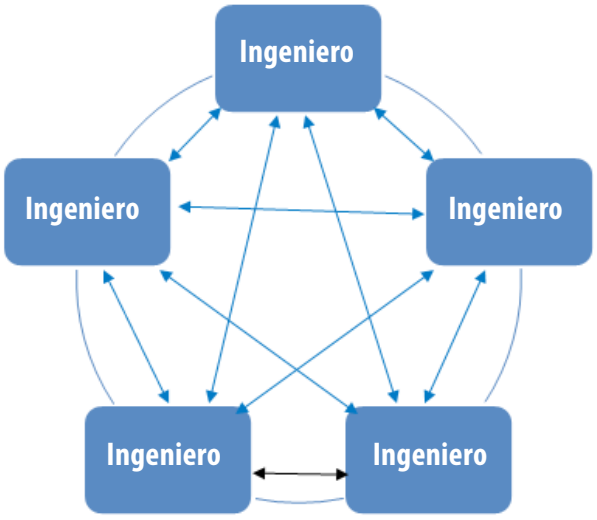


Figura 2
Fuente: Propia.

Los integrantes del equipo tienen la misma autoridad, lo que permite aumentar la motivación de cada uno, frente a sus actividades. El inconveniente es que se generan dificultades a la hora de resolver diferencias, pues no hay una cabeza líder y la toma de decisiones se hacen por mayoría, lo que puede presentar inconformidades.

Este tipo de organización puede funcionar correctamente, en equipos pequeños, cuyos integrantes estén acostumbrados a trabajar en equipo.

Organización horizontal

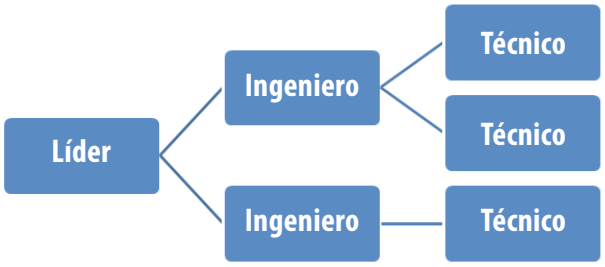


Figura 3
Fuente: Propia.

Todos los integrantes tienen la misma relevancia, excepto el líder, quien es el encargado de incentivar la participación de cada uno. Si todos tienen las mismas capacidades, cada uno puede asumir una responsabilidad en un área específica (implementación, diseño, calidad, etc.).

Gestión del riesgo

Un riesgo es cualquier hecho que puede afectar en forma negativa el proceso de desarrollo de un proyecto. Si se identifican estos riesgos durante el proceso, es posible prevenir sus consecuencias o cambiar su enfoque para minimizarlas.

La gestión del riesgo implica identificar, eliminar o minimizar cada una de los posibles riesgos. Esto puede ser llevado a cabo por un integrante del equipo que hace la labor de coordinador de riesgos, pero todos deben adoptar una «mentalidad de riesgo» para identificarlos en todo momento del proceso.

Tipos principales de riesgos

- Evitables: por ejemplo, que alguno de los integrantes del equipo decide retirarse. En este caso puede corregirse o mitigarse para continuar el proceso.

- No evitables: por ejemplo, cuando se cambian las condiciones o requerimientos, después que han sido desarrolladas. Este tipo de riesgos pueden ser beneficiosos, pues el proyecto podría no continuarse y así no seguir gastando recursos innecesariamente.

Actividades principales en la gestión de riesgos

- Identificación.
- Planificar su eliminación.
- Priorizar los riesgos para su eliminación.
- Eliminación o atenuación.

Estas actividades pueden ser plasmadas en una tabla que se muestra a continuación y que representa un plan de trabajo para la gestión de los mismos.

	Probabilidad (1-10) (1= menor probab.)	Impacto (1-10) (1= menor impacto)	Costo eliminar (1-10) (1= Menor costo)	Cálculo prioridad (11-Pr)*(11-Im)*C	Prioridad (Menor número mas prioritario)
La prioridad mas alta	10 (muy posible)	10 (mayor impacto)	1 (Menor costo)	(11-10)*(11-10)*1	1
La prioridad mas baja	1 (poco posible)	1 (menor impacto)	10 (Mayor costo)	(11-1)*(11-1)*10	1000

Nº riesgo	Nombre riesgo	Probabilidad (1-10)	Impacto (1-10)	Costo (1-10)	Prioridad	Plan Eliminación	Ingeniero resp.	Fecha eliminación (meta)
1	Superposición de imágenes	3	10	1	8	Experimentar con imágenes en Java	Ines García	30/2/04
2	¿Existen Drivers de Java?	9	6	8	80	Explorar la existencia de las mismas en la WEB	Rosa Fernández	28/2/04
3	Juan López puede salir del proyecto	3	7	9	288	Rosa Fernández inspeccionará todos los trabajos de Juan	Rosa Fernández	Continuo

Imagen 1
Fuente: http://www.ctr.unican.es/asignaturas/Ingenieria_Software_4_F/Doc/M2_08_Administracion-2011.pdf

Una vez identificados los riesgos, se le asigna un puntaje de acuerdo a diferentes factores, como prioridad, costo, impacto, etc. Esto dependerá del tipo de proyecto que se esté desarrollando. Tener organizados los riesgos ayuda a tener una mejor visión del impacto que pueden causar y en esta medida darles el tratamiento necesario

Algunas fuentes de riesgos

- Que los líderes del proyecto no tengan el compromiso suficiente.
- Que el cliente o usuario no tenga el compromiso suficiente.
- Requisitos mal comprendidos.

- Mala participación del usuario o cliente en el proceso.
- Que el producto final no cumpla las expectativas del cliente o usuario final.
- Que cambie el alcance, objetivos o requerimientos.
- Que las aptitudes o conocimientos del personal no sean suficientes.

Herramientas online para gestión de proyectos

Con el avance que ha tenido internet y la cobertura que los gobiernos han venido implementando en las ciudades, ha aumentado también la tendencia del trabajo colaborativo online. Y es aquí donde las herramientas online para gestión de proyectos juegan un papel muy importante.

A continuación relacionaremos algunas funcionalidades de este tipo de herramientas:

- Intercambiar archivos en tiempo real.
- Enviar o recibir notificaciones instantáneas de reuniones, cambios, etc.
- Control y seguimiento de los plazos establecidos.
- Seguimientos de costes.
- Gestión de la relación con clientes o usuarios del proyecto.
- Actividades de facturación.
- Creación de wiki y versiones de desarrollo.
- Chat y videoconferencias.

Dentro de las muchas opciones que existen en la red, se han seleccionado 10 herramientas que por sus características, se resaltan de las demás. A cada una se le ha incluido el link respectivo del sitio oficial para que usted conozca sus características específicas:

- ACTIVE COLLAB ActiveCollab
- ASSEMBLA Assembla
- BASECAMP Basecamp
- CENTRAL DESKTOP Central Desktop
- CONFLUENCE Confluence
- KAPOST Kapost
- PRODUCTEEV Producteev
- TEAMBOX Teambox
- TEAMLAB TeamLab
- TIMEDOCTOR Time Doctor

Paradigmas de la Ingeniería de software

Modelo en cascada o clásico (modelo tradicional)

La característica principal es que para poder avanzar a una siguiente etapa del ciclo de vida del software, se debe haber finalizado la etapa inmediatamente anterior. La metodología que sigue el modelo en cascada es:

Figura 4
Fuente: Propia.

Si por ejemplo, se detecta algún error de diseño, se debe rediseñar y realizar una nueva programación, lo que aumenta los costos de desarrollo.

Modelo en espiral (modelo evolutivo)

En un modelo orientado a riesgos, en el que el proyecto se divide en proyectos más pequeños y en cada uno de ellos se identifican los riesgos más importantes a fin de controlarlos. Se van desarrollando versiones incrementales de cada uno, hasta obtener el producto final.

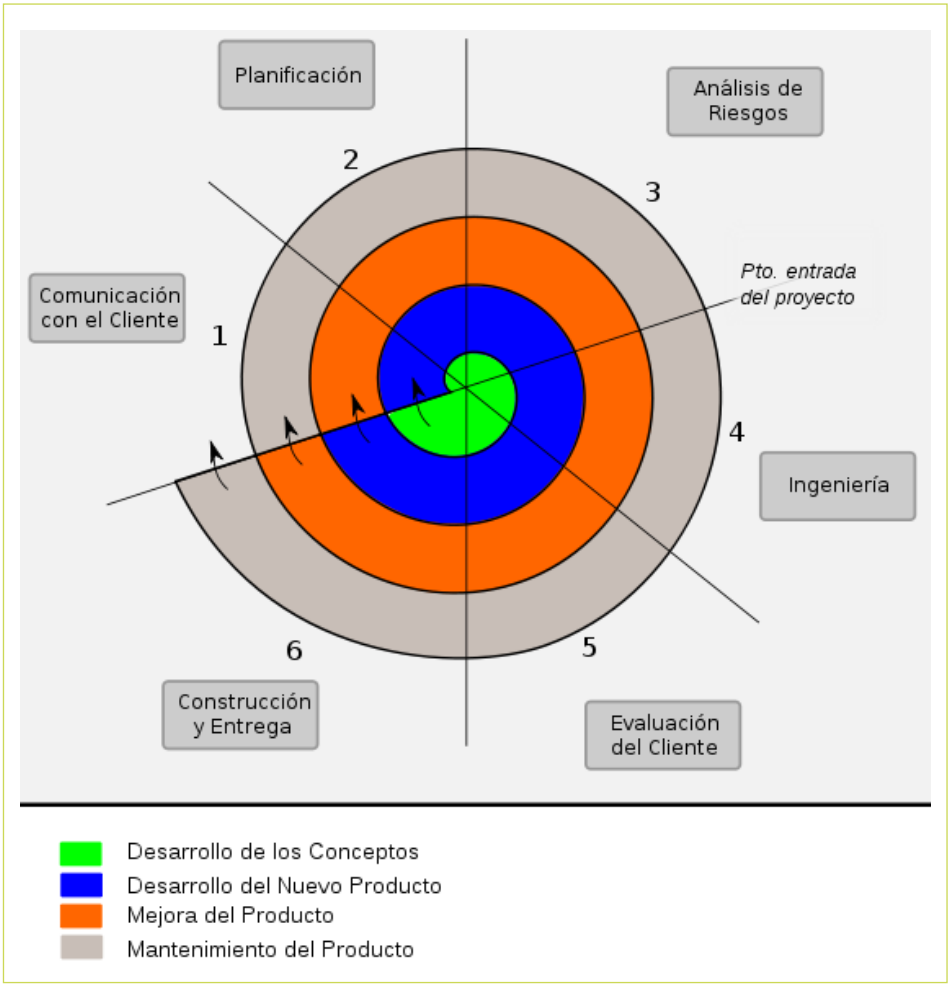


Imagen 2
Fuente: http://1.bp.blogspot.com/_S5W0bf4uECM/TL5uOoqW4MI/AAAAAAAAABI/wk8pbHQALHU/s1600/espiral2.png

Modelo de prototipos

Permite al usuario evaluar el producto desde las primeras etapas de desarrollo e interactuar con el equipo de trabajo, para ir verificando el cumplimiento de los requerimientos exigidos.

El modelo sigue la siguiente metodología:

- Definir objetivos globales.
- Identificar requisitos conocidos.
- Identificar áreas que requieren mejor definición.
- Diseño rápido de prototipo y modelado.

RAD (Rapid Application Development)

Busca reducir los riesgos segmentando el proyecto para facilitar cualquier cambio que se requiera. Hace uso de la iteración por prototipos en cada una de las etapas de desarrollo, involucrando al usuario en todo momento y utilizando herramientas de software, para así generar productos de alta calidad y en el menor tiempo posible.

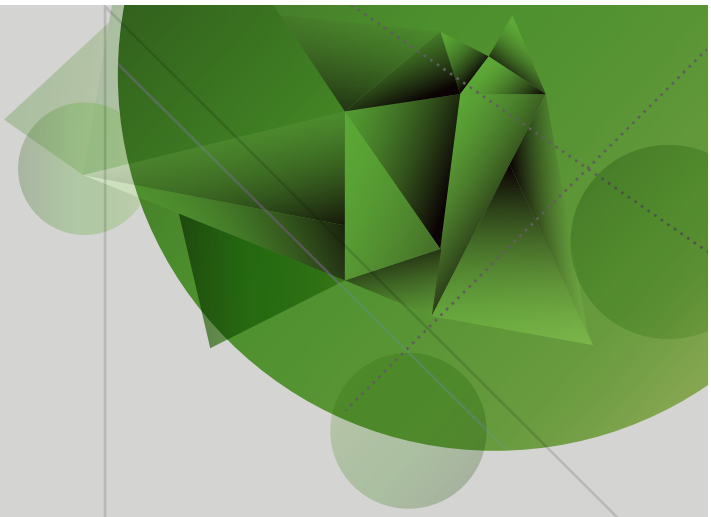
Las herramientas de software pueden ser:

- Generadores de Interfaz gráfica de usuario (GUI).
- Computer Aided Software Engineering (CASE).
- Sistemas de gestión de bases de datos (DBMS).
- Lenguajes de programación de cuarta generación.
- Generadores de código.
- Técnicas orientadas a objetos.

2

Unidad 2

Introducción a la ingeniería de requisitos



Ingeniería de software

Autor: Fredy Alonso León Socha

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

La formación del Ingeniero en sistemas involucra el conocimiento de dispositivos físicos encargados de soportar el procesamiento y transmisión de la información, los cuales se convierten en elementos fundamentales al momento de desarrollar una solución en estos campos.

El conocimiento y manejo adecuado de estas tecnologías permite resolver problemas de sistemas, usando el soporte de soluciones computacionales e informáticas eficientes aplicando el conocimiento científico.

De esta forma, la presente asignatura es fundamental para el estudiante que se inicia en esta profesión, ya que le permite visualizar y conocer, los fundamentos de la formación en tecnología, su evolución histórica, los campos de acción, y las principales herramientas que utiliza un ingeniero en sistemas. De igual manera, este curso aporta las bases para reconocer y entender los diversos dispositivos físicos relacionados con su objeto de estudio, así como la labor que estos cumplen en la implementación de soluciones en el campo de los sistemas, de tal forma que puedan realizar la mejor elección, en el momento de desarrollar un proyecto de apropiación o desarrollo.

U2

Metodología

Para el desarrollo de los temas teóricos del curso se asignarán a los estudiantes algunas actividades de investigación y exposición, actividades que serán guiadas y apoyadas por el tutor y complementarán los contenidos publicados en plataforma. Las evaluaciones de las actividades serán utilizadas como retroalimentación para los expositores y para el mismo grupo.

El tutor realizará una teleconferencia semanal de un tema especifico, en el cual se presentarán ejemplos para reforzar dicho tema. En cada semana se publica el material de estudio que apoyara el desarrollo de los contenidos temáticos correspondientes a dicha semana, ya sea en recursos bibliográficos, archivos digitales o referencias electrónicas (páginas web).

Se recomienda el ingreso diario a la plataforma para aprovechar al máximo el curso.

Introducción a la ingeniería de requisitos

Concepto de requisito

Es una funcionalidad que un sistema debe realizar para que cumpla con el objetivo para el cual fue desarrollado.

Es diferente un requisito a un deseo del cliente.

La misión del Ingeniero de software es transformar las solicitudes del cliente o usuarios en requerimientos.

Deben ser verificables mediante análisis, inspección, demostración o pruebas.

Deben ser necesarios, por lo que omitirlos causarían un mal funcionamiento del sistema.

Deben ser consistentes en el sentido que no debe contradecir otro requerimiento.

Ejemplo: interfaces graficas, control de errores, estándares utilizados, capacidad de rendimiento, seguridad de los datos, que sea portable, usabilidad, etc.

A continuación se muestra una grafica que representa como se encuentran organización los requisitos según (Pohl, 1997).

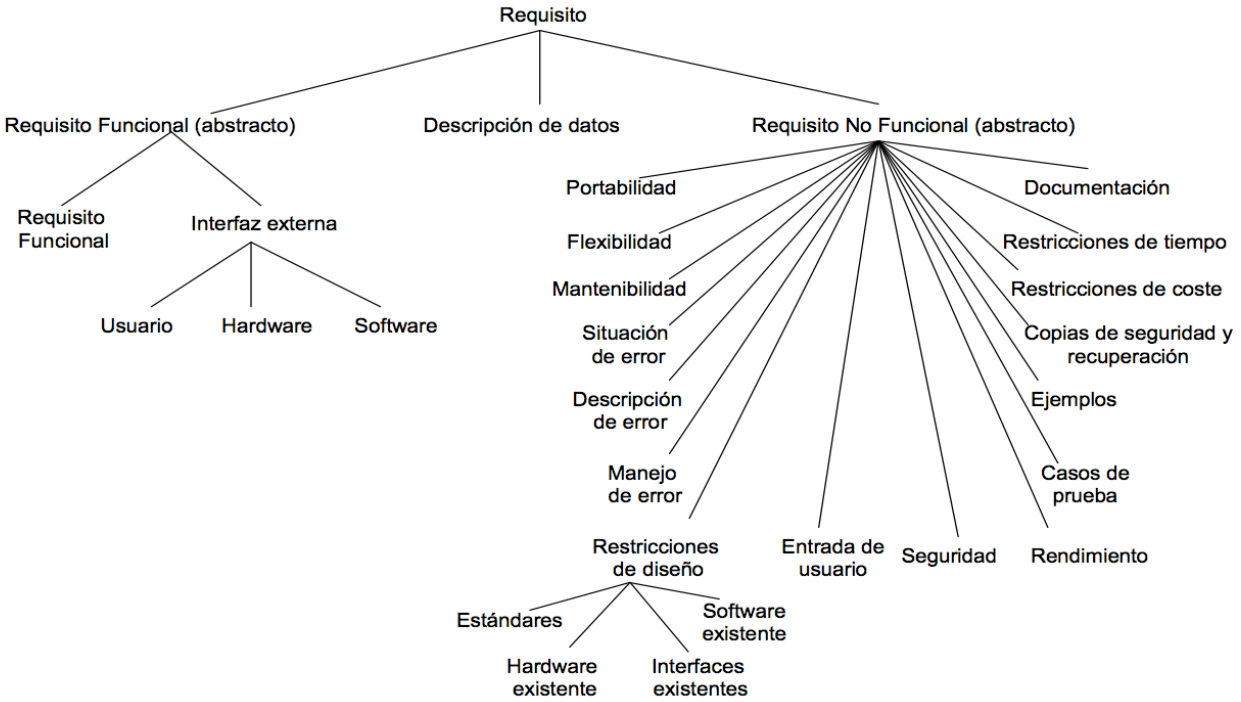


Imagen 1
Fuente: http://ocw.usal.es/ensenanzas-tecnicas/ingenieria-del-software/contenidos/Tema3-IntroduccionalaIR-1pp.pdf

Requisitos de información

Hacen una descripción de los datos que el sistema debe almacenar y procesar para poder cumplir los objetivos para los cuales fue desarrollado.

Ejemplo: se deberá almacenar información referente a préstamos realizados en el banco, tales como nombre, tipo de préstamo, cantidad, fecha del préstamo, fecha finalización, domicilio del cliente, etc.

Requisitos de interfaz

Definen la forma en que interactuara con otros sistemas.

Ejemplo: el sistema deberá establecer comunicación bluetooth con la impresora portátil para realizar la impresión del recibo. Los datos deben ser enviados en formato XML.

Requisitos funcionales

Representan aquellas tareas especificas que el sistema debe realizar y como deberá ser el comportamiento frente entradas o situaciones particulares.

Ejemplo: el usuario deberá poder buscar los artículos por nombre o código.

Requisitos de funcionalidad

Permiten especificar como deberá ser el comportamiento de un sistema.

La herramienta más utilizada para el levantamiento de este tipo de requisitos son los Casos de Uso. Estos describen simbólicamente como es la interacción de los diferentes actores que intervienen en el sistema y las acciones que realizan sobre el mismo sistema. Todas las acciones deben obtener un resultado que sea observable y de valor para cada uno de los actores.

Los casos de uso hacen parte del Lenguaje Unificado de Modelado UML. Este tema será tratado más profundamente en la semana 4. La siguiente tabla resume los elementos que componen un diagrama de casos de uso:

Nombre	Función	Representación grafica
Actor	Role que realiza un usuario dentro de un sistema.	
Casos de Uso	Acción específica que se realiza después de una orden de una entrada externa.	
Relaciones de Uso, Comunicación y Herencia	Asociación: indica la invocación desde un actor o caso de uso a otra operación (caso de uso).	
	Dependencia o Instanciación. Indica cuando una clase depende de otra, es decir, se instancia (se crea).	
	Generalización: es exclusivo para casos de uso. Puede ser de Uso (<<uses>>) o de Herencia (<<extends>>). extends: se usa cuando un caso de uso tiene características similares a otro. uses: se usa cuando varios caso de uso tienen un conjunto de características similares.	

Cuadro 1
Fuente: Propia.

Para comprender un poco más el tema, a continuación encontrara una imagen se muestra un modelo de casos de uso para un sistema de compra de obras de arte.

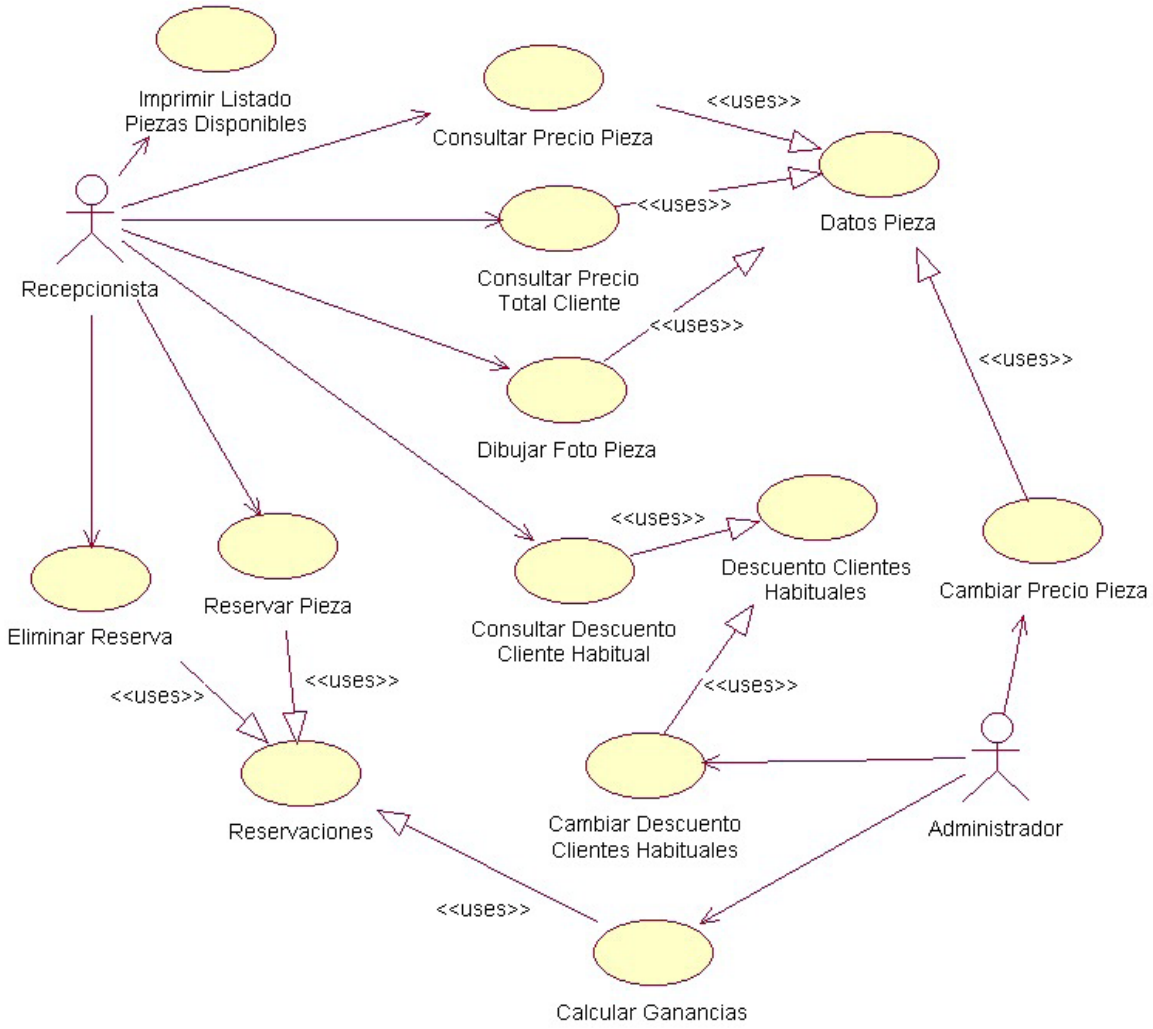


Imagen 2
Fuente: http://www.geocities.ws/vidalreyna/usecase

Requisitos de datos

Se refieren a los tipos de datos que manejara el sistema con el fin de determinar la forma en que deben ser procesados.

Ejemplo: fechas, enteros, float, string.

Requisitos no funcionales

A continuación se relacionan algunas características que tienen los requisitos no funcionales:

Se encuentran por fuera de la forma en que debe funcionar el sistema.

Se asocian con restricciones que tendrá un sistema, ya sean internas o externas. Una restricción es una limitante que hace que el problema se pueda resolver de una forma determinada ej.; que al final de un proceso haya que imprimir automáticamente un documento en una impresora con unas características especificas.

La tarea de verificación normalmente es difícil de realizar.

Hacen una definición de las funcionalidades globales que tendrá el sistema.

Se relacionan con aquellas propiedades adicionales de un sistema.

A continuación se muestra un mapa conceptual referente a requisitos no funcionales:

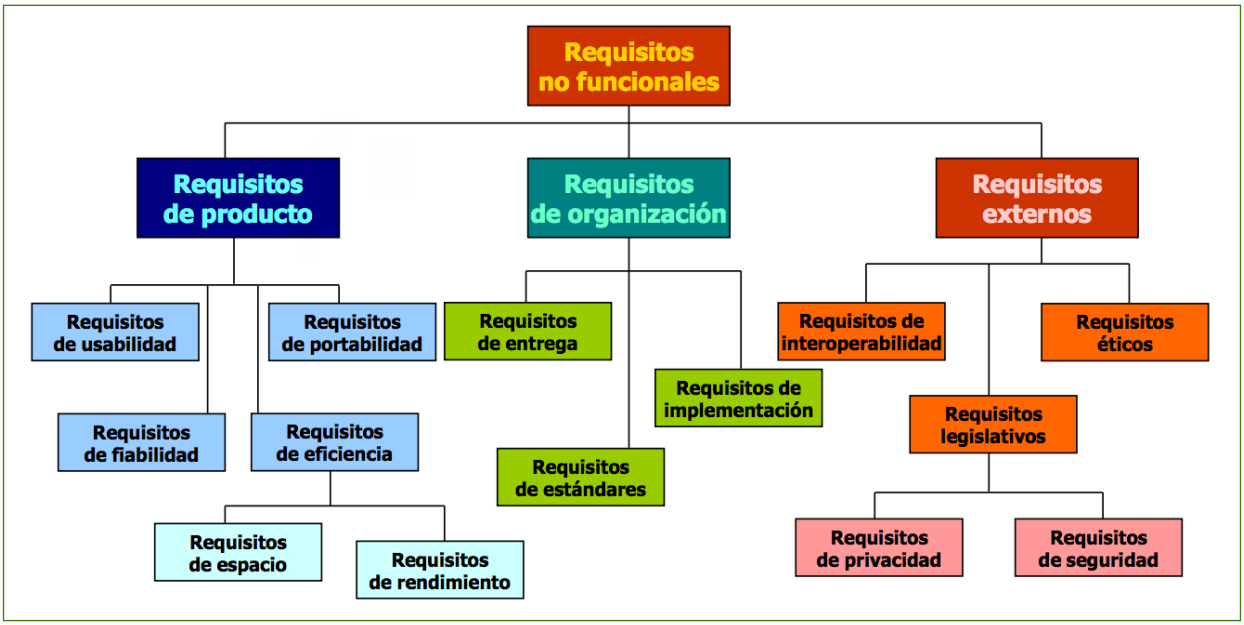


Imagen 3
Fuente: http://vignette3.wikia.nocookie.net/requerimientos-software/images/3/30/CLASIFICACION_DE_REQUERIMIENTOS_NO_FUNCIONALES.png/revision/latest?cb=20140430011055&path-prefix=es

Requisitos de seguridad

Nivel de protección de los datos que almacena el sistema, para uso no autorizado, perdidas de información, accesos no autorizados etc.

Requisitos de usabilidad

La forma en que puede ser usado por el usuario.

Ejemplo: Se usara por personas con discapacidad auditiva.

Para medir la usabilidad se toman algunos puntos de referencia tales como:

Tiempo de capacitación para que un usuario puedan manipularlo correctamente.

Tiempos para realizar tareas específicas.

Interfaces amigables para el usuario.

Documentación de uso, tipo y forma de presentación, configuraciones, ayudas, etc.

Requisitos de mantenimiento

Reflejan la facilidad, periodicidad, y costos que deben tenerse en cuenta para realizar las tareas de mantenimiento, con el fin de corregir defectos, hacer mejoras, etc.

Requisitos de comprobabilidad

El nivel en el que un sistema permite ser comprobado, verificado o medido, en cuanto a las funciones que realiza

Requisitos de disponibilidad

Relacionan el tiempo total que el software está disponible y operable para su uso, tomando como referencia un patrón de tiempo.

Ejemplo: el servidor de hosting debe tener el 99.9% de disponibilidad online.

Requisitos de escalabilidad

Grado en el que el sistema puede aumentar sus capacidades. Por ejemplo: Aumentar el número de conexiones o usuarios conectados.

Requisitos de extensibilidad

Miden la capacidad del sistema para incrementar las funcionalidades.

Ejemplo: que adicional a conectarse a otros dispositivos mediante wifi, que pueda conectarse vía bluetooth.

Otros requisitos no funcionales

Requisitos de entorno, de presentación, requisitos culturales, requisitos legales.

Objetivos de la ingeniería de requisitos

Identificar y documentar los que los clientes y usuarios necesitan.

Crear la documentación necesaria donde se describa el comportamiento externo y restricciones de un sistema, de tal forma que cumpla las necesidades identificadas.

Analizar y validar la documentación de requisitos para determinar que sean viables, que estén completos y consistentes.

Realizar la devolución de necesidades.

Conceptos de ingeniería de requisitos

A continuación se relacionaran algunas definiciones planteadas por especialistas en esta rama:

Según (Reifer, 1994), "Es el uso sistemático de procedimientos técnicas, lenguajes y herramientas para obtener con un coste reducido el análisis, documentación, evolución continua de las necesidades del usuario y la especificación del comportamiento externo de un sistema que satisfaga las necesidades del usuario".

Según (Loucopoulus y Karakostas, 1995), "Es un proceso sistemático de desarrollo de requisitos mediante un proceso cooperativo consistente en analizar el problema, documentar las observaciones resultantes en una variedad de formatos de representación, y comprobar la exactitud de la comprensión conseguida".

Actividades básicas de la Ingeniería de requisitos

Para completar el proceso adecuadamente es necesario realizar una especificación y administración de cada uno de los requi-

sitos de los clientes o usuarios. Es por esto que para lograr este objetivo se siguen cuatro etapas fundamentales:

Extracción

Se enfoca al descubrimiento de requisitos del sistema.

Análisis

Se analiza el documento levantado en la etapa de extracción. Se realiza una lectura, conceptualización, investigación de los requisitos, se identifican problemas y soluciones .También se programan reuniones con el equipo de trabajo y con el cliente para discutirlos.

Especificación

Se escribe el documento formal de requisitos en forma detallada. Aquí se hace uso de técnicas como UML y otros estándares de documentación.

Validación

Se enfoca en la verificación de los requisitos planteados en el documento final, de tal forma que realmente hagan una representación valida de lo que el sistema debe realizar. Las validaciones son internas porque debe verificarse lo que se hace y externas porque deben ser aprobadas por el cliente o usuario.

Análisis y negociación de requerimientos

Objetivos del análisis de requisitos

- Realizar una detección temprana de problemas.
- Construir un modelo de software que condense los requisitos del sistema.
- Asegurar viabilidad técnica, de costes y planificación.

Tareas del análisis de requisitos

Reconocimiento del problema

Evaluar las situaciones planteadas por el cliente a fin de identificar la necesidad existente y se realiza un plan de trabajo. También se inicia un proceso de comunicación con el equipo de trabajo a fin de reconocer y analizar grupalmente el problema a solucionar.

Evaluación y síntesis

Analizar, evaluar y condensar el documento de requerimientos, detallando las funciones del sistema, interfaces, diseño, etc.

Modelado

Utilizar técnicas de modelado como UML, Casos de Uso, para definir roles y funciones de cada uno de los actores que intervienen en el sistema, partiendo de los objetivos y requisitos documentados

Especificación

Para dar una representación del programa que pueda ser revisada y aprobada por el cliente. Se realiza la documentación

Revisión

En esta etapa normalmente se generan cambios de comportamiento, funciones establecidas para el sistema, la forma en que se va a presentar la información, criterios de validación y se reevalúa globalmente el plan de proyecto a fin de reafirmar las proyecciones realizadas durante la etapa de análisis.

Gestión de cambios en los requisitos

Busca evaluar y planificar los cambios al proyecto de una forma eficiente, de tal forma que se pueda asegurar la calidad y continuidad del servicio. Debe existir una comunicación con asertiva con los demás proceso.

Roles que interactúan en la gestión de cambios

La siguiente tabla resume los roles principales que intervienen en el procedimiento de control de cambios.

Rol	Descripción
Comité de control de cambios (GCC)	Se encarga de aprobar o rechazar solicitudes de cambios. Conformado por clientes y Desarrolladores.
Promotor del cambio	Quien lidera y hace genera la solicitud para un cambio de requisitos.
Evaluador	Se encarga de analizar el impacto que puede causar la petición de cambio en el sistema, ya sea a nivel técnico, de cliente, de marketing.
Modificador	Se encarga de realizar el cambio solicitado, de acuerdo al análisis que previamente ha sido revisado y aprobado.
Verificador	Se encarga de verificar si los cambios se han hecho en forma correcta.
Validador	Normalmente el cliente final, quien es el que hace una validación del cambio realizado.

Cuadro 2
Fuente: Propia.

Diagrama del proceso

La siguiente figura muestra las actividades que se desarrollan durante la gestión de un cambio de requerimientos de un proyecto de software.

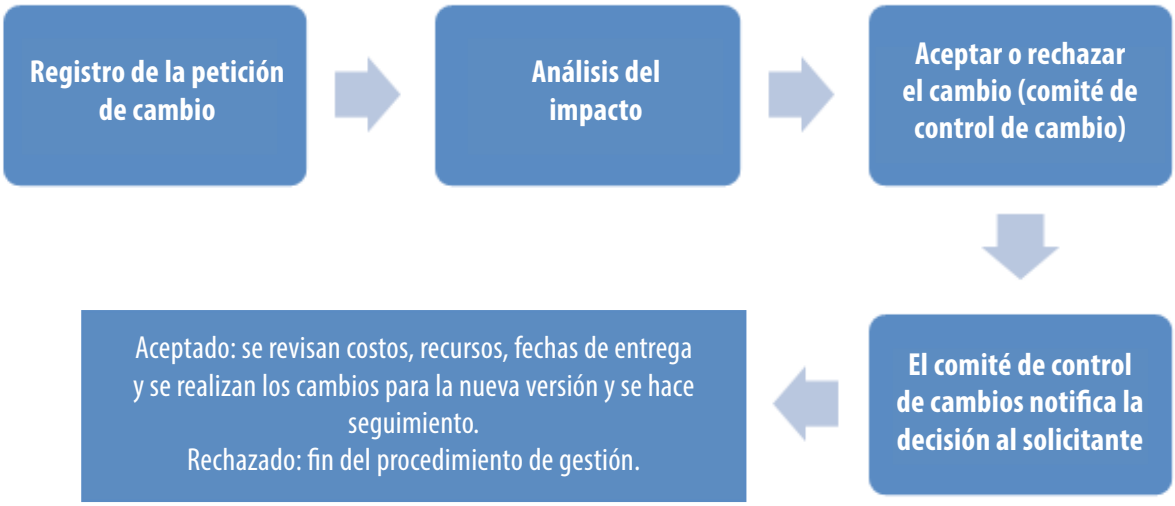


Figura 1
Fuente: Propia.

Flujos de entrada y salida

A continuación se muestra las diversas entradas y correspondientes salidas que se requieren para la gestión de cambios por parte del Comité de Control de Cambios.

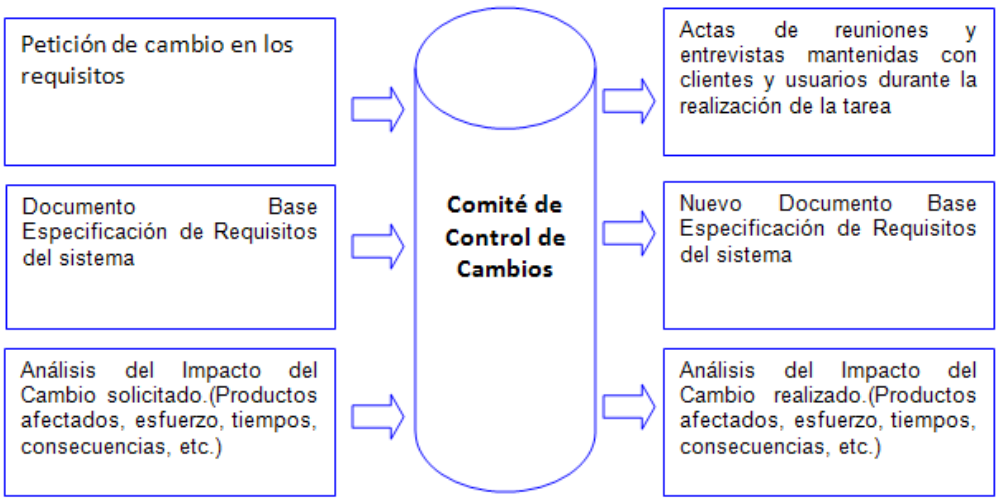


Imagen 4
Fuente: Propia.

Proceso de negociación

La tarea principal es la de realizar una discusión de los problemas encontrados y buscar una solución que sea benéfica para los usuarios y personas involucradas en el desarrollo del proyecto.

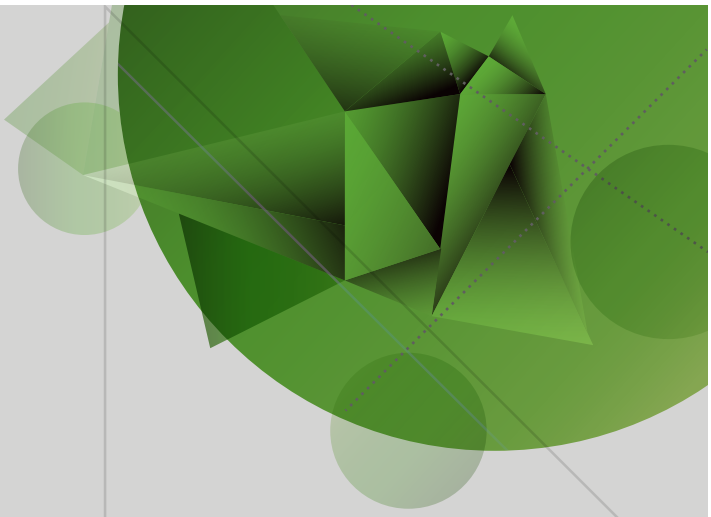
Durante este proceso se llevan a cabo 3 tareas importantes:

1. Discutir los requisitos que presentan conflictos.
2. Establecer una prioridad para cada uno.
3. Definir un compromiso final sobre los mismos.

2

Unidad 2

Especificación y validación



Ingeniería de software

Autor: Fredy Alonso León Socha

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

Como Ingenieros de sistemas es muy importante conocer el hardware asociado a los sistemas computacionales, así como la labor que desempeñan cada uno de los componentes, dentro de dichos sistemas. Esta unidad se ha diseñado para cumplir ese objetivo, por lo que usted conocerá la configuración básica de un computador, los puertos disponibles para conexión de periféricos y los dispositivos más comunes que se conectar al computador, ya sea como entradas o salidas.

Conociendo estos elementos usted podrá plantear soluciones, frente a la configuración básica o fallas comunes en el hardware del sistema de cómputo o en los elementos conectados al mismo.

U2

Metodología

El módulo se basa en la lectura individual de la presente cartilla y posterior desarrollo de actividades planteadas en la plataforma online.

Se espera una participación y acceso a plataforma a su consideración, como mejor pueda organizarse, para cubrir los objetivos de aprendizaje y participación. Por mi experiencia, recomiendo la asistencia diaria para aprovechar al máximo el curso. Es interesante seguir los hilos de participación 2-3 veces al día y dedicar uno momento para analizar y participar inteligentemente.

La comunicación conmigo la pueden realizar abiertamente a través del Foro y personalmente a través del email.

Al final de cada sección enviaré un breve mensaje resumen destacando los mejores en sus intervenciones individuales y en los trabajos de grupo de esa sección.

Aquellos que hayan llamado mi atención negativamente por su falta de participación o por lo inadecuado de las mismas les enviaré un mensaje privado.

Especificación y validación

Proceso de validación de requisitos

Busca garantizar que los requisitos cuenten con las características de calidad suficientes para poder continuar a la siguiente fase del proceso de desarrollo. Algunas de estas características son:

- Que sean consistentes.
- Completitud.
- Precisos.
- Acordes con la realidad.
- Que puedan ser verificados.

El proceso también busca disminuir el riesgo de tener que corregir después que el proyecto ha avanzado, lo que generaría altos costos.

- Como se muestra en la figura 1, la validación de requisitos recibe como entradas: el documento de requisitos (creado durante la fase de especificación).
- Los estándares o lineamientos por los que deben regirse.
- El conocimiento que el equipo de trabajo encargado de realizar la validación.

Una vez realizado el proceso, como productos finales se tendrán: Un listado de problemas y un listado de acciones que permiten dar solución a dichos problemas y mejorar el documento de requisitos.

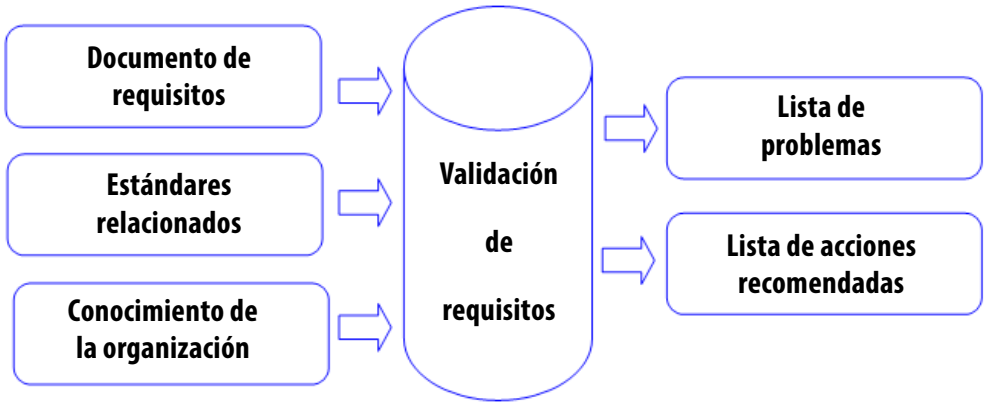


Figura 1.Entradas y salidas del la validación de requisitos
Fuente: Propia.

Técnicas de validación de requisitos

Prototipado

El prototipado busca construir un modelo usable de un producto de software, de tal que clientes o usuarios puedan realizar pruebas y determinar las correcciones o mejoras que se deberán realizar, teniendo en cuenta la especificación de requisitos que dicho producto debe tener.

Existen diferentes tipos de prototipos, los cuales se muestran a continuación:

Mock-ups: se enfoca en hacer una representación de la interfaz grafica de la aplicación y las pruebas se limitan a este contexto.

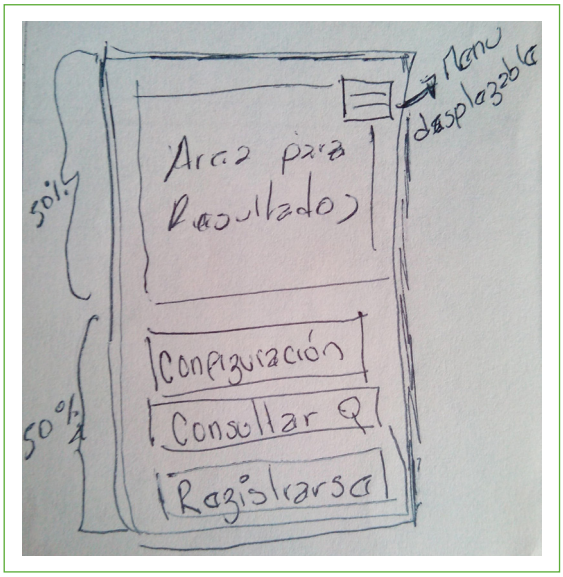


Imagen 1
Fuente: Propia.

Storyboards: muestra gráficamente diferentes tareas que debe realizar la aplicación.

Imagen 2
Fuente: Propia.

Maquetas: se crea un prototipo funcional, con las pantallas propuestas para la interfaz gráfica, botones, conexiones entre pantallas. Dependiendo de la fidelidad que se requiera, se puede programar acciones básicas para mostrar resultados concretos del sistema.

Independientemente del tipo de prototipo que se utilice, se deben seguir una serie de pasos para realizar el proceso de validación de requisitos, como se muestra en la siguiente figura.

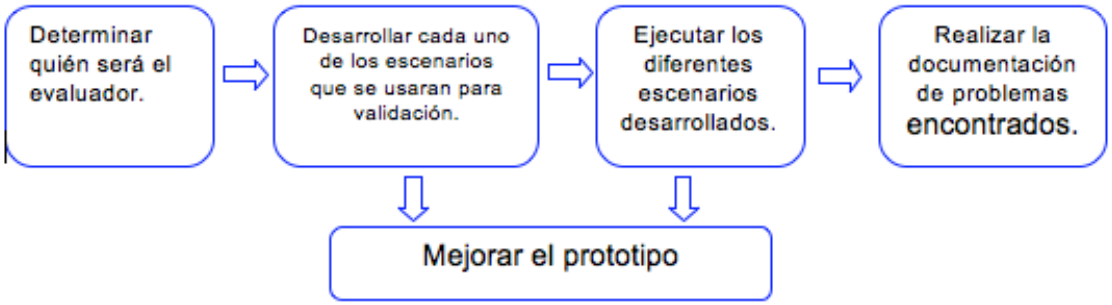


Figura 2
Fuente: Propia.

Generación de casos de prueba

También se le llama revisión de requisitos o Test de requisitos. Este método se centra en la verificabilidad, una de las características de la calidad del software. Se definen unos casos de prueba de tal forma que se pueda comprobar cada uno de los requisitos funcionales del software.

Para que se pueda cumplir la verificabilidad de un requisito debe ser posible definir uno o varios casos de prueba.

A continuación se desarrollara un ejemplo de desarrollo de un caso de prueba para el siguiente requisito:

X. El sistema deberá generar un informe.

Para este caso se define la tarea que debe realizar el software, mas no el cómo lo debe realizar, ni que resultados deberían generarse. Esto pasa porque el requisito no está bien definido y algunas razones son las siguientes:

- No se indica cuales son los datos base para generar el informe.
- ¿El informe debe ser semanal, entre fechas, de un pedido, por un cliente, de ventas, etc.?
- ¿En qué momento del proceso se debe generar?
- ¿Quién lo genera, automáticamente o manualmente?

Teniendo en cuenta lo anterior, el caso de prueba seria:

1. Se seleccionarán los datos A, B, C, para generar el informe. Estos datos ya deben estar bien definidos, con sus respectivos datos (cantidades, fechas, detalles, etc.).

2. Se deberá invocar la función de generar informe (Automática o Manual).
3. Se generará un informe con los datos A, B, C.…indicando además fecha y hora de generación. Se debe incluir en la parte superior, el titulo del informe, el cual es ingresado por el usuario, etc.

Creación de manuales de usuario

Lo que busca es que se pueda crear el manual de usuario, partiendo de las especificaciones de requisitos. De no poderse lograr, seguramente dicha especificación no es suficientemente detallada.

Animación y validación de modelos

Consiste en realizar animaciones de los modelos de pantallas del sistema y demás especificaciones, para que el cliente o usuario pueda visualizar como quedara el producto final.

Problemas en la determinación de requisitos

Relacionados con los usuarios

- No hay claridad en lo que quieren.
- Falta de vinculación en el proceso de elaboración de requisitos.
- Tienden a agregan nuevos cambios, después de haber definido costes y cronogramas de trabajo.
- Baja comunicación entre ellos y el equipo de trabajo.
- Falta de participación en revisiones.
- No hay comprensión de los problemas técnicos que generan.
- No hay comprensión del proceso de desarrollo.

Relacionados con el área de análisis

- Falta de homogeneidad y claridad en la redacción del documento de requerimientos.
- Demasiado uso de condicionales.
- Poco uso de terminología propia de la aplicación.

Relacionados con el personal de desarrollo

- Diferentes terminologías usadas entre los ellos y los usuarios.
- Desarrollar sistemas basados en modelos existentes, dejando a un lado las necesidades reales del cliente.
- Dejar el análisis de requisitos en manos de personas que no cuentan con el perfil adecuado para este tipo de actividades, con el fin de disminuir costos.

Gestión de requisitos y herramientas

Procedimiento para revisión de requisitos

El siguiente es el procedimiento normal que se debe ser a la hora de hacer la revisión del documento de especificación de requisitos:

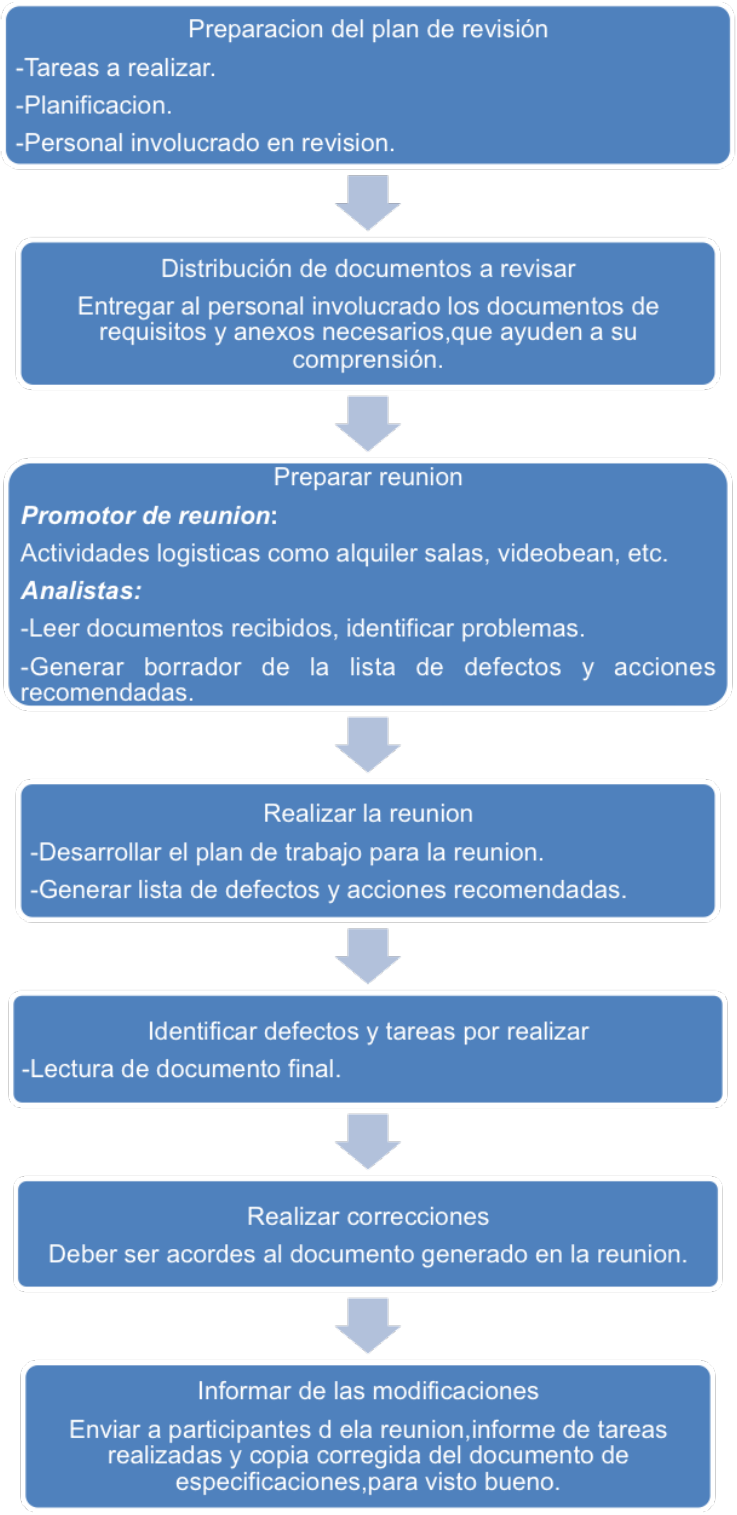


Figura 3
Fuente: Propia.

Gestión de cambios de los requerimientos

A continuación se muestra un diagrama del proceso general que se sigue en la gestión de requisitos.

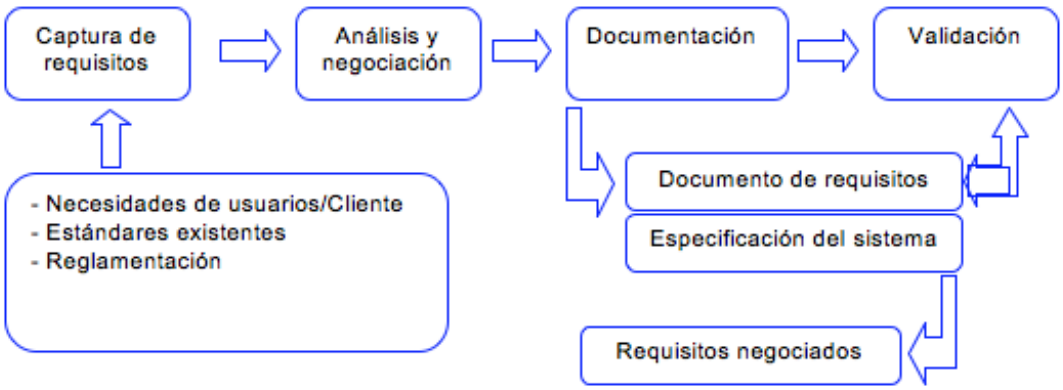


Figura 4
Fuente: Propia.

El proceso inicia con una fase de exploración que busca conocer las necesidades que tienen los usuarios o clientes. También se debe documentar el contexto en el que se desenvolverá el proyecto.

Continua con la Captura de requisitos, que es cuando se crea un documento de requerimientos, los cuales pasaran a una etapa de Análisis para determinar problemas y una etapa de Negociación, ya que seguramente se deberá llegar a un acuerdo con el cliente, teniendo en cuenta que habrá requisitos imposibles de cumplir, por tiempo, tipos de tecnología, costos, etc.

Definidos los requerimientos finales, se pasa a la etapa de documentación, cuyo producto final es un documento con lista definitiva de requerimientos, que pasaran a una etapa de validación para realizar ajustes de calidad. Finalizada esta etapa se tendrá entonces la lista definitiva y corregida de requisitos, listos para que el área de desarrollo comience su trabajo.

Proceso de negociación

En esta etapa se identifican los requisitos que presentan conflicto y se discuten con el cliente, con el fin de llegar a un acuerdo que beneficie a ambas partes. Los conflictos se generan a raíz que en el proyecto están involucrados diferentes personas, cada uno con sus propios intereses.

En esta etapa se deben desarrollar las siguientes tareas:

Realizar una discusión de los requisitos que presentan conflicto

Definir el nivel de prioridad para cada uno de los requisitos que presentan conflicto.

Eliminar, combinar o modificar los requisitos necesarios

Establecer el compromiso final al que se llego con el cliente

Herramientas comerciales

Rational Requisite Pro

- Desarrollado por IBM.
- Los requisitos son almacenados en forma de documentos.
- Gestiona el control de cambio de requisitos, para especificaciones de software y pruebas.
- Usa Oracle como motor de base de datos, corriendo sobre Unix o Windows.
- También soporta SQL Server sobre Windows.
- Página oficial: http://www.ibm.com/developerworks/ssa/library/IBM_Rational_RequisitePro.html

CaliberRM

- Permite la captura y comunicación de las necesidades del usuario.
- Permite adaptarse a procesos de negocio.
- Ofrece variedad de soportes para clientes finales: Navegadores Web, Eclipse, Microsoft Visual Studio y Windows.
- Ofrece un repositorio seguro y centralizado de acuerdo a las necesidades del proyecto.
- Facilita el trabajo colaborativo.
- Permite el análisis de impacto Trazabilidad en tiempo real.
- Página oficial: http://www.borland.com/en-GB/Products/Requirements-Management/Caliber

IRqA (Integral Requisite Analizer)

- Brinda soporte para Captura, Analisis, Especificacion y validación de requerimientos.
- Gestiona el estado de arte.
- Los requisitos son almacenados en documentos MS Word.

Telelogic Doors

- Está diseñado para realizar tareas de captura y enlace de requisitos.
- Analizar y gestionar cambios a la información.
- Ligado a especificaciones técnicas de proyectos a los requisitos y normas especificas.
- Permite mejorar los procesos de comunicación, colaboración y validación.
- Página oficial: http://telelogic-doors.software.informer.com/

Visure Requirements

- Soporta captura manual de requisitos, servicios/soluciones y casos de prueba.
- Permite importar requisitos desde MS Word, Excel o XRI.
- Permite definir tipos de requisitos funcionales y no funcionales.
- Gestiona los cambios durante las fases del ciclo de vida.
- Gestiona la trazabilidad de requisitos y su respectivo ciclo de vida.
- Página oficial: http://www.visuresolutions.es/
- CASE Spec
- Gestión , especificación y análisis de requisitos

- Trabajo colaborativo
- Análisis de trazabilidad
- Diseño de especificaciones y documentación
- Testeo y validación
- Página Oficial: http://analysttool.com/

Herramientas libres

REM (Requisite Management)

- Brinda el soporte necesario para las fases de Ingeniería de Requisitos de proyectos de software.
- Creado por el profesor Amador Durán Toro, de la universidad de Sevilla
- Permite agregar objetivos, actores.
- Crear casos de uso.
- Crear diferentes tipos de requisitos: Funcionales, no funcionales, de restriccion, de almacenamiento de información.
- Genera matrices de rastreabilidad.
- Página oficial: https://www.lsi.us.es/descargas/descarga_programas.php?id=3

DRES

- Basado en PHP.
- La administración del proyecto se hace mediante navegadores web.
- La documentación ese genera en formato HTML.
- Usa MySQL como motor de base de datos y CORBA como servidor para almacenamiento y gestión de requerimientos.
- Soporte para extensiones en XML, directorios jerarquicos, resportes, etc.
- Página oficial: http://sourceforge.net/projects/dres/

OSRMT (Open Source Requeriments Management Tool)

- Permite la trazabilidad completa del Ciclo de vida de desarrollo de software: características, gestión de requisitos, diseño, implementación y pruebas.
- Control de versión de documento de requisitos.
- Página oficial: http://sourceforge.net/projects/osrmt/

Heler

- Monousuario
- Soporte para Windows y licencia GPL (GNU, 2008).
- Creada bajo el Modelo Vista Controlador.
- Escrito en JAVA.
- Usa PostgreSQL como motor de base de datos.
- Usa Poseidon para modelado de diagramas UML.
- Módulos contenidos: Proyecto, stakeholder, actores y casos de uso, requisitos.

Herramientas en la nube

Gather Space

- Gestor de requerimientos.
- Casos de uso e historial de usuario.
- Jerarquía y asociaciones.
- Casos de prueba.
- Seguimiento de validaciones.
- Página oficial: http://www.gatherspace.com

ITestMan

- Permite generar un repositorio de la información que se genera duran te el proceso de gestión de requisitos.

- Puede generar automáticamente las especificaciones de requisitos así como la documentación de pruebas de validación.
- Generar informes en formatos diferentes y para cada una de las áreas creadas para el proyecto.
- Acceso a la información en tiempo real, respecto a avances de requisitos, pruebas, problemas etc.
- Está ligado con estándares de desarrollo como MIL-STD 498, ISO 12207, ED109, CMMI nivel 3, etc.
- Página oficial: http://www.polar-consultores.es

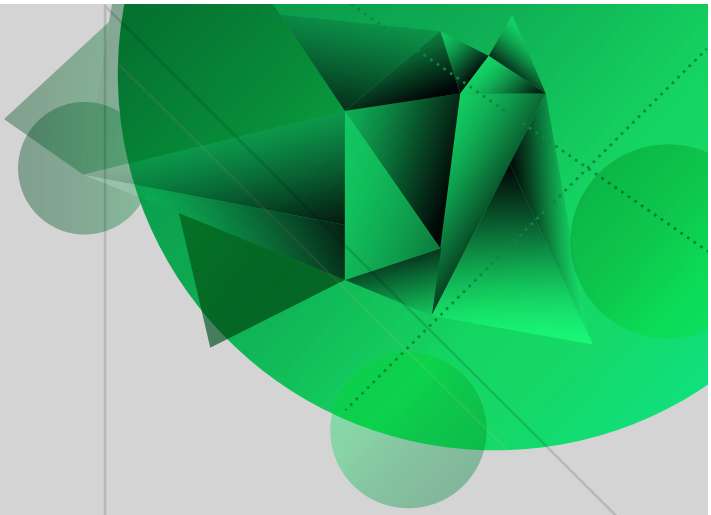
Accept software

- Permiten recopilar, organizar y requisitos del producto, ideas, estrategias y directorios.
- Uso de componentes modulares.
- Anticipación de problemas y conflictos.
- Priorización y reducción de tiempos basada en información real de mercado.
- Toma de decisiones en los diferentes niveles.
- Página oficial: http://www.accept360.com/

3

Unidad 3

Diseño orientado a objetos de sistemas de software



Ingeniería de software

Autor: Fredy Alonso León Socha

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

El Ingeniero de sistemas debe estar preparado para realizar diseños e implementaciones de redes cableadas e inalámbricas, por lo que debe conocer los medios disponibles para la transmisión de datos, sus características, equipos y todos los conceptos relacionados con esta temática. En la presente cartilla encontrará la fundamentación teórica relacionada con la implementación de redes cableadas e inalámbricas, que le permitirá tener las bases suficientes para enfrentarse a un diseño de red en la vida practica.

U3

Metodología

El módulo se basa en la lectura individual de la presente cartilla y posterior desarrollo de actividades planteadas en la plataforma online.

Se espera una participación y acceso a plataforma a su consideración, como mejor pueda organizarse, para cubrir los objetivos de aprendizaje y participación. Por mi experiencia, recomiendo la asistencia diaria para aprovechar al máximo el curso. Es interesante seguir los hilos de participación 2-3 veces al día y dedicar uno momento para analizar y participar inteligentemente.

La comunicación conmigo la pueden realizar abiertamente a través del Foro y personalmente a través del email.

Al final de cada sección enviaré un breve mensaje resumen destacando los mejores en sus intervenciones individuales y en los trabajos de grupo de esa sección.

Diseño orientado a objetos de sistemas de software

Un objeto se compone de datos y procedimientos que representan una entidad. Para el objeto también se define las diferentes formas de interacción con otros objetos, lo que comúnmente se la llama la "Interfaz del objeto".

Teniendo en cuenta lo anterior, el DOO se centra en la creación de objetos e interacciones que estos tienen entre si y que permiten cumplir las funciones establecidas para un sistema. De esta forma la información y el procesamiento se modularizan, permitiendo incluir las tres características en las que se basa cualquier método de diseño: Abstracción, Ocultamiento de información y Modularidad.

Conceptos del diseño orientado a objetos

Objetos, operaciones y mensajes

Objeto: es la asociación de un elemento del mundo real con su rol dentro del software. Por ejemplo; en un sistema de pedidos, un objeto podría ser el usuario, el administrador o un elemento de información.

Los objetos encapsulan su estado y la forma en que están representando la información y la comunicación con otros objetos se hace a través del envío de parámetros

Operaciones: métodos o servicios. Se refiere a las acciones que puede realizar el objeto. Estas pueden cambiar las propiedades de los objetos o iniciar un evento con nueva información para otro objeto del sistema.

Mensajes: solicitudes hechas a objetos para que ejecuten operaciones o procedimientos y devuelvan información. Cuando se envina mensajes a un objeto que tiene una parte privada, se está ocultando la información y los detalles de implementación del objeto, ya que el único acceso son los datos que se le envían al objeto y el resultado que este devuelve a quien le envío el mensaje.

Metodologías de desarrollo de software

OOHDM

OOHDM - *Object Oriented Hypermedia Design Method*.
Esta enfocada al desarrollo hipermedia y web.
El proceso de desarrollo se centra en 4 fases:
Modelado conceptual.
Diseño navegacional.
Diseño abstracto de interfaz.
Implementación.

Estas actividades se realizan en una mezcla de estilo incremental, iterativo y basado en prototipos de desarrollo.

Los modelos orientados a objetos se construyen en cada paso que mejora los modelos diseñados en iteraciones anteriores y consta de las siguientes fases:

Fase Conceptual. Construcción de esquema conceptual, el cual se representa por objetos, relaciones y colaboraciones entre ellos. El esquema conceptual se constituye de clases, relaciones y subsistemas.

Fase Navegacional. Centrada en establecer los modelos de navegación del sistema y se expresa por un lado mediante Clases navegacionales: Nodos, enlaces y estructuras de acceso y por el otro, mediante el esquema de contextos navegacionales, que es un conjunto de nodos, enlaces, clases de contextos, y otros contextos navegacionales (contextos anidados), los cuales se definen por comprensión o extensión, o por enumeración de sus miembros.

Fase de Diseño de interfaz abstracta. Define los objetos de interfaz con los que el usuario va a interactuar, cuáles de estos objetos van a intervenir en los procesos de navegación y como se va a llevar a cabo la sincronización de los objetos multimedia y el interfaz.

Fase de Implementación. Los modelos se llevan a lenguaje de programación, con el fin de obtener el ejecutable de la aplicación. Utilizando las herramientas y Mecanismos que ofrece el lenguaje de programación seleccionado.

SOHDM

Diseño en Escenarios Orientados a Objetos en Hipermedia (Scenario - based Object-oriented Hypermedia Design Methodology). La primera fase del ciclo de vida que comienza con la creación de los escenarios como técnica de elicitación y definición de requisitos, mediante la utilización de DFD. Enfocado a aplicaciones web.

La siguiente imagen muestra las fases que sigue la metodología SOHDM:

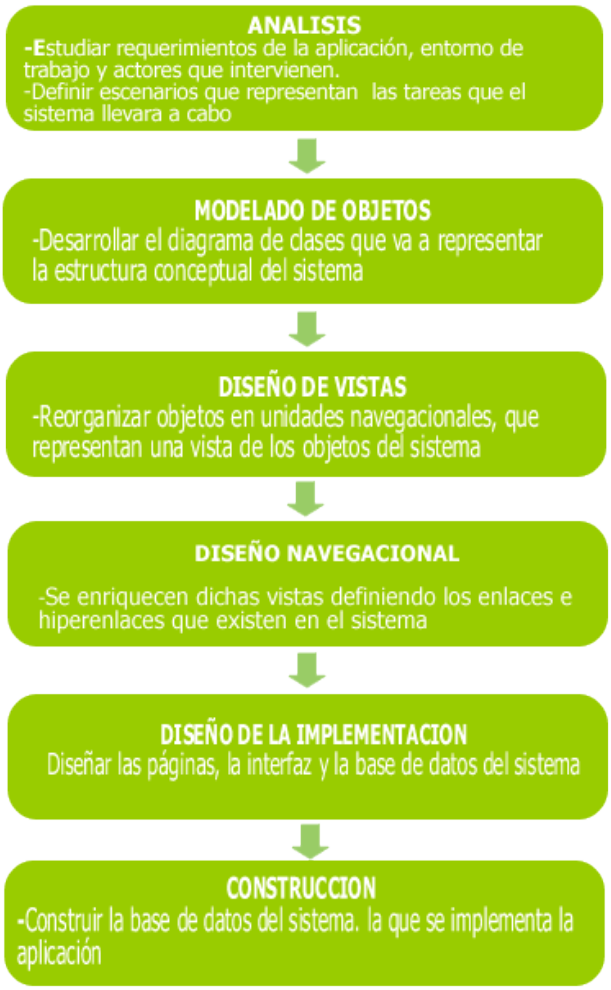


Imagen 1
Fuente: Propia.

WSDM

- Método de Diseño para Sitios Web (Web Site Design Method).
- El usuario hace parte del diseño de los objetos de información, de acuerdo a los requisitos de información para el sitio web.

- El diseño Navegacional se cimenta en 3 capas: Contexto, Navegación e Información.

Las fases de la metodología WSDM se presentan a continuación:

Imagen 2
Fuente: Propia.

RNA

- Relationalship Navigational Analisys o Método de Análisis de navegación relacional.
- Hace énfasis en las tareas de análisis de entorno y elementos de interés en la fase de especificación de requisitos.
- Los requisitos conceptuales s y navegacionales se analizan en forma independiente.
- Usada principalmente para proyectos web.

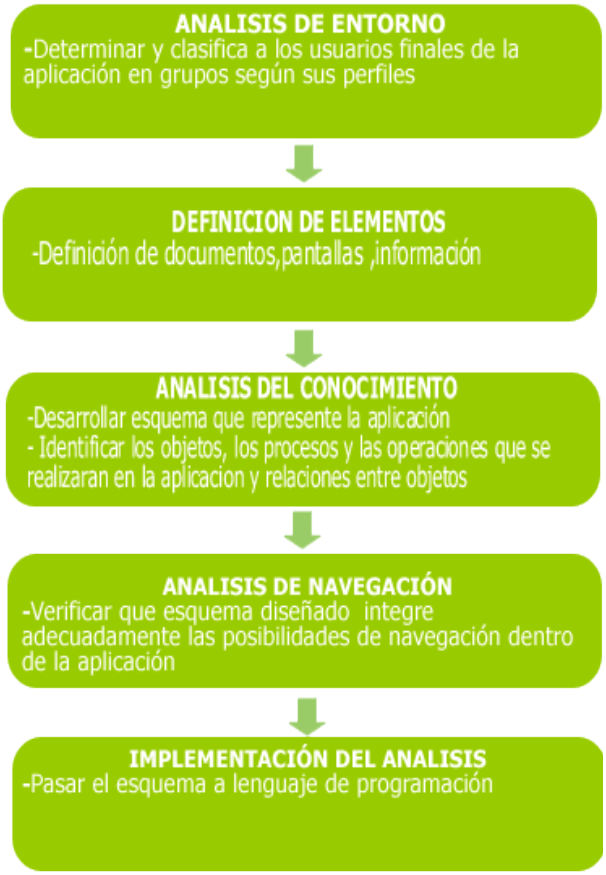


Imagen 3
Fuente: Propia.

UML

- UML - Unified Modeling Language, es un lenguaje grafico para hacer modelos independientemente de los métodos de análisis y diseño.
- Permite representar, visualizar, especificar, construir y documentar los elementos de un sistema de software, como el diseño, comportamiento, arquitectura, etc.

UML sigue las siguientes fases:

Imagen 4
Fuente: Propia.

UML maneja el concepto de vistas, entendida como un conjunto de diagramas que juntos muestran las funcionalidades del sistema. Las vistas que maneja UML son:

Vista use-case: muestra las funcionalidades desde el punto de vista de actores externos.

Vista lógica: muestra el diseño de funcionalidad interna del sistema, teniendo en cuenta su estructura estática y conducta dinámica.

Vista de componentes: muestra cómo están organizados los componentes de código.

Vista concurrente: muestra los aspectos de comunicación y sincronización.

Vista de distribución: muestra la arquitectura física del sistema.

Las partes que componen el UML son:

- Las vistas.
- Los diagramas.
- Los elementos del modelo.
- Lo mecanismos de extensión.

NOTA: esta metodología será tratada con más detalle en la semana 6.

Modelos de proceso y ciclos de vida.

Modelo fuente.

Permite el desarrollo de sistemas en forma iterativa y encapsulada.

Las etapas que sigue son las siguientes:

1. Planificación: es independiente de la metodología orientada a objetos
2. Construcción: esta se divide en: Planificación, Investigación, Especificación, Implementación y Revisión
3. Entrega: es independiente de la metodología orientada a objetos.

Estas fases se desarrollan en 2 periodos definidos que son:

- Crecimiento: tiempo durante el cual se hace el diseño del sistema.
- Madurez: tiempo durante el cual se desarrollan las actividades de mantenimiento y mejora del producto.

La siguiente grafica muestra el ciclo de vida que sigue el Modelo Fuente:

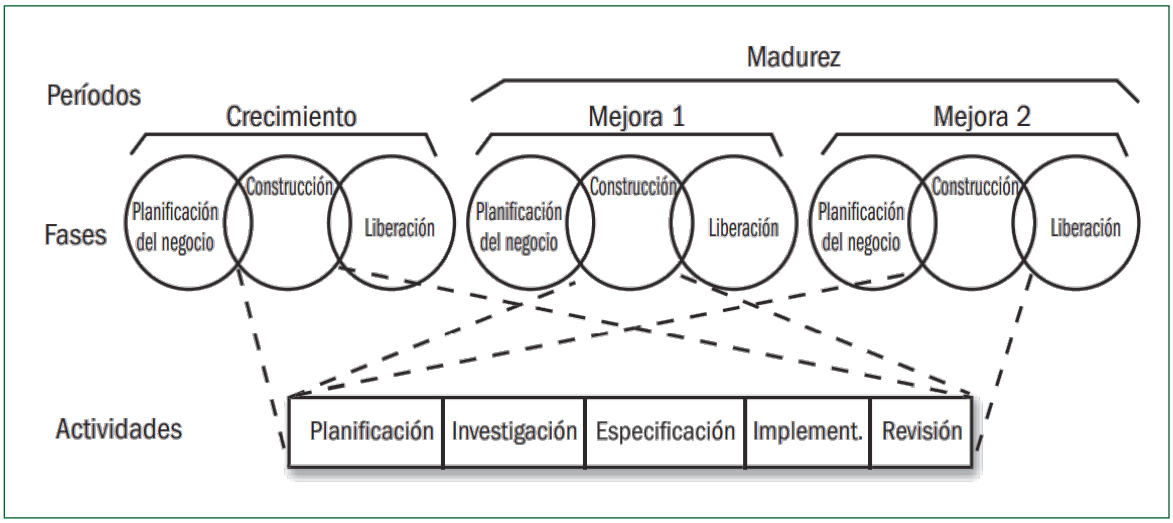


Imagen 5
Fuente: http://3.bp.blogspot.com/-ys_bK8WaRrA/T61OUjmFm1I/AAAAAAAAALo/DlQBW-8QZio/s1600/Orientado+a+Objetos.png

Modelo de agrupamiento

- El conjunto de clases son separadas de acuerdo al objetivo para la cual fueron creadas, formando así diferentes clúster de clases.
- Cada clúster tiene sus propios subciclos de vida que pueden anidarse durante el periodo de desarrollo. Todos los subciclos de vida siguen las mismas fases: Especificación, Diseño, Implementación, Verificación/Validación y Generalización.

La siguiente imagen representa como es la agrupación en el modelo de agrupamiento:

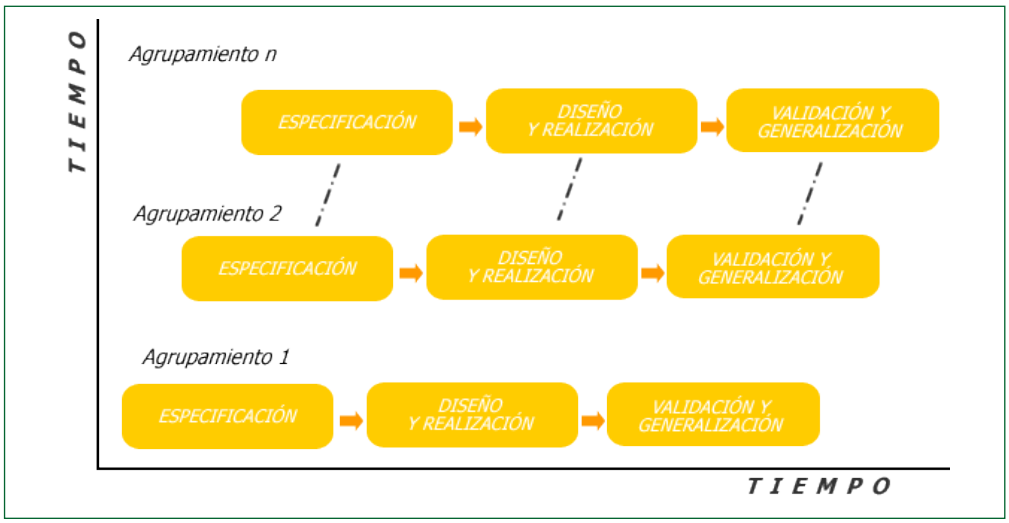


Imagen 6
Fuente: Propia.

Modelo remolino

Maneja el concepto de dimensiones, las cuales pueden organizarse en forma diferente, de acuerdo a las necesidades y tipo de proyecto las cuales se relacionan a continuación:

Figura 1
Fuente: Propia.

Modelo pinball

Relaciona el juego de Pinball con la forma en que se desarrolla el proyecto. Quien juega es el Equipo de desarrollo, la pelota es el Proyecto y los obstáculos a sobrepasar son las clases, atributos, métodos, relaciones, colaboraciones, herencias, agregaciones y sub-sistemas. Estos obstáculos se pueden sobrepasar en cualquier orden, lo importante es sobrepasarlos correctamente.

Sobrepasando poco a poco los primeros obstáculos, el jugador se aproxima a la parte final del juego, que son las fases de programación y pruebas. Para jugar se plantean dos estilos diferentes: El primero es el Modo Seguro, que es usando tecnologías y métodos que ya han sido probados y el segundo, el Método Limite, que es aventurarse a utilizar nuevas tecnologías y métodos.

La pelota siempre va a estar moviéndose entre los obstáculos por lo que el paso de los mismos se hace en forma iterativa y está en la Habilidad del equipo de desarrollo sobrepasarlos correctamente y en el menor tiempo posible.

La siguiente imagen muestra como seria la implementación del método Pinball:

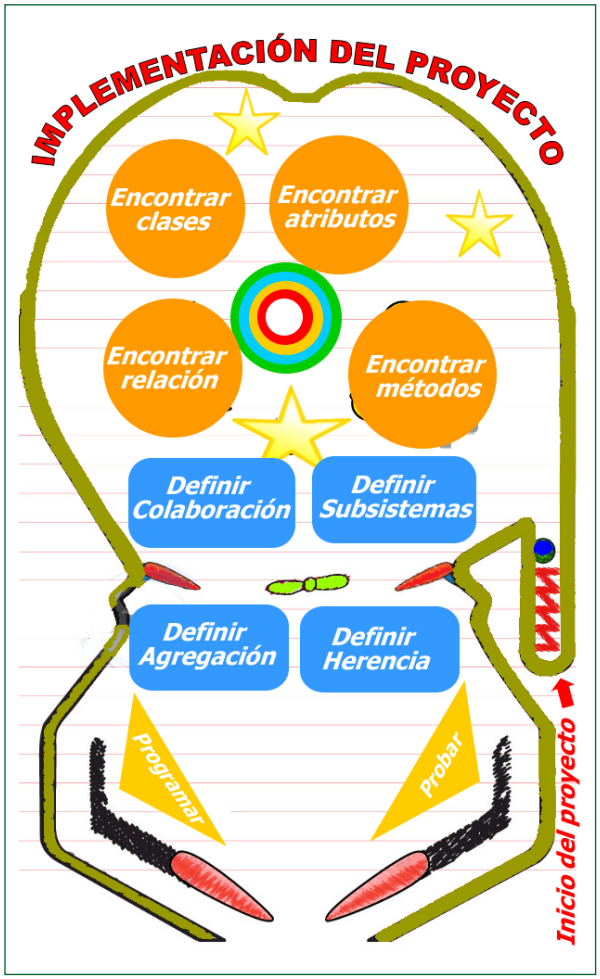


Imagen 7
Fuente: Propia.

Metodologías ágiles

Scrum

- Para cuando se requieren productos finales en plazos cortos de desarrollo.
- Cuando los requisitos del proyecto cambian regularmente.
- Cuando se quieren tener equipos de trabajo.
- Proyectos en los que se requiere de altos grados de innovación, competitividad, flexibilidad y productividad.
- Cuando se ha extendido demasiado el tiempo de entrega del producto final, se han generado altos costos de desarrollo y la calidad no es la que se esperaba.
- El equipo trabaja en forma colaborativa para minimizar riesgos.

El proceso Scrum se muestra a continuación:

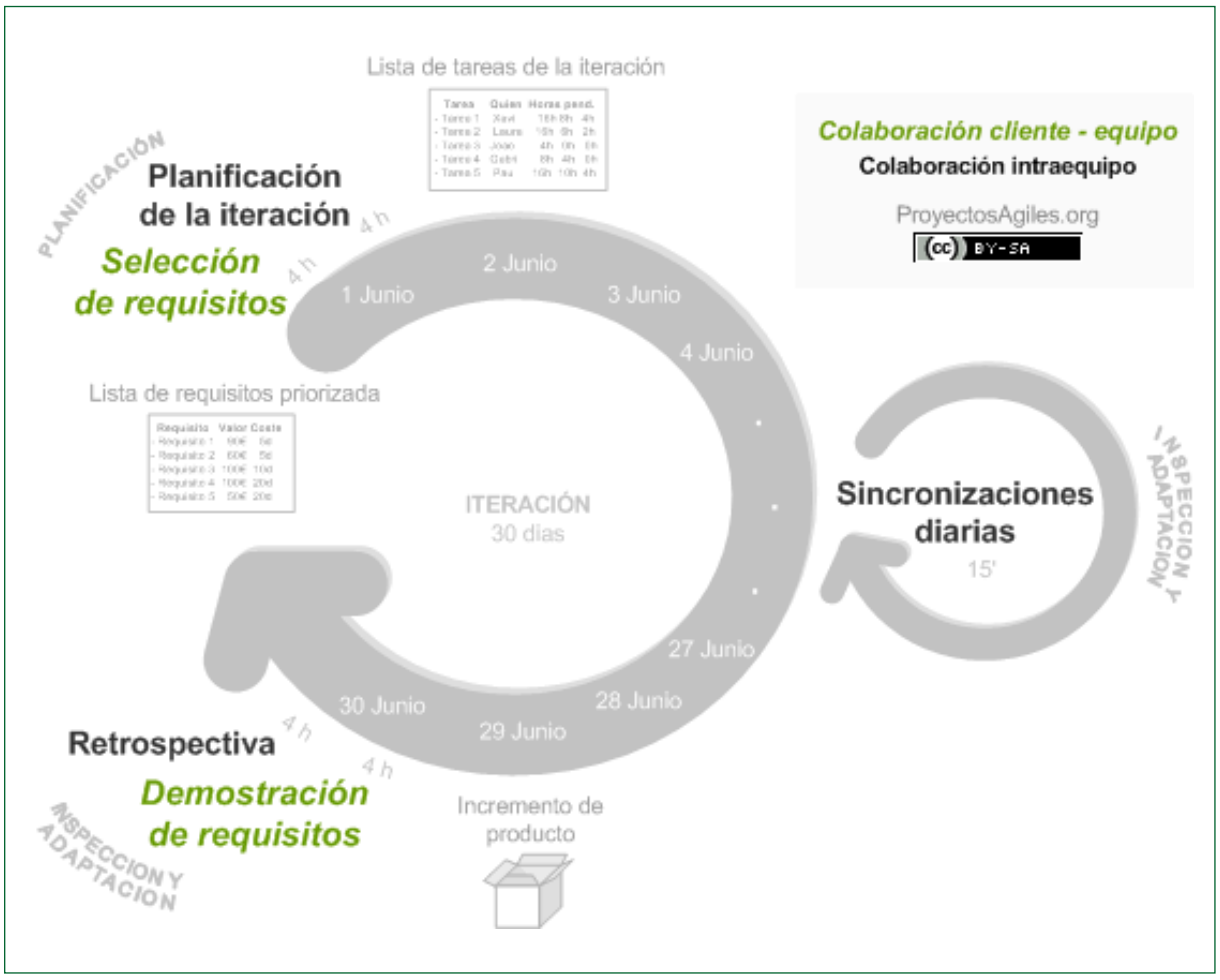


Imagen 8
Fuente: https://scrumenespanol.files.wordpress.com/2015/09/diagrama-proceso-scrum.gif?w=553&h=414

La siguiente tabla muestra un resumen de los roles que existen en Scrum:

Roles principales	
Product Owner	Representa el punto de vista del cliente.
ScrumMaster (o facilitador)	Asegurar que el proceso de desarrollo se lleve a cabo sin distracciones y bajo los parámetros, reglas y condiciones establecidos.
Equipo de desarrollo	Su fin es el de entregar el producto final. Equipo interdisciplinario con habilidades de análisis, diseño, desarrollo, pruebas, documentación, etc.

Roles secundarios	
Administradores (managers)	Establece las condiciones de ambiente de trabajo para el desarrollo.
Stakeholders	Involucrados alrededor del proyecto: Clientes, Proveedores, Vendedores, etc.

Tabla 1
Fuente: Propia.

RAD

RAD (Rapid Application Development) o Desarrollo Rápido de Aplicaciones, comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE, a usabilidad, utilidad y la rapidez de ejecución. Se enfoca a que el ciclo de vida del desarrollo sea lo más corto posible.

La siguientes son las fases que sigue la metodología RAD:

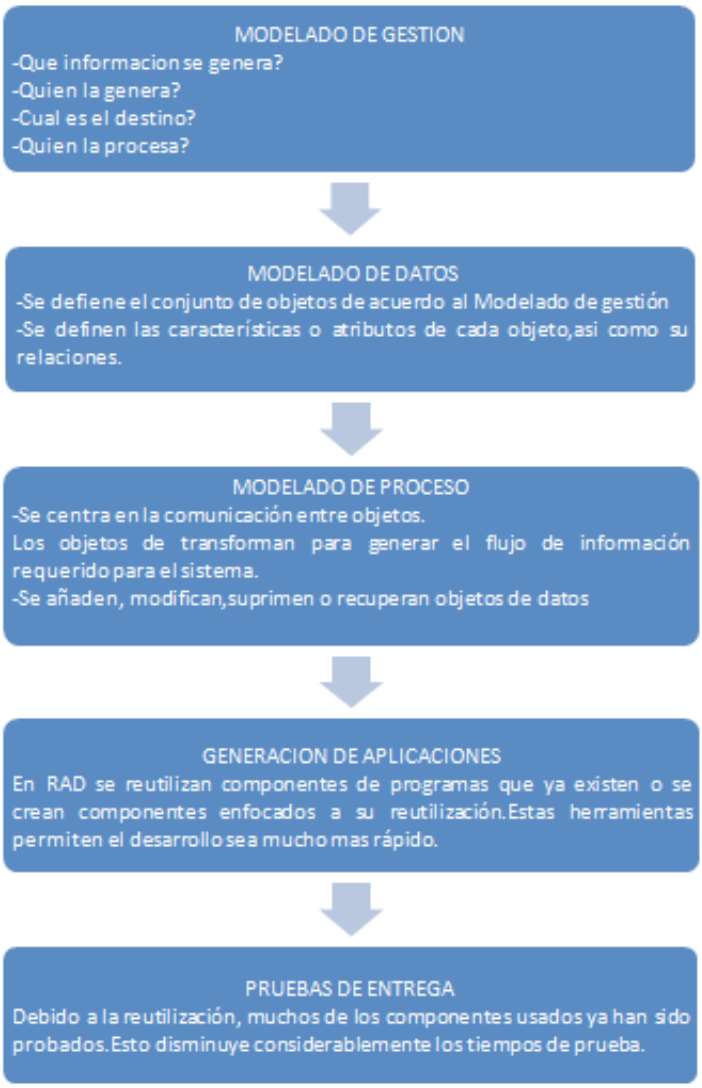


Imagen 9
Fuente: Propia.

Dentro de las características principales se puede señalar:

Se compone de equipos híbridos: normalmente lo conformas 6 personas. Cada desarrollador debe tener capacidades de analista, diseñador y programador.

Hace uso de herramientas especializadas: Por ejemplo para crear prototipos, multilenguaje, herramientas colaborativas, librerías, Apis, Calendario de trabajo, etc.

Timeboxing: todo lo que no sea esencial se elimina con el fin de cumplir el calendario propuesto.

Prototipos Iterativos. Se hacen reuniones JAD (Join Application Development), donde se reúne el cliente y desarrolladores, a fin de generar el documento inicial de requisitos. De ahí en adelante se realizar iteraciones hasta finalizar el producto, siguiendo el esquema que se muestra a continuación:

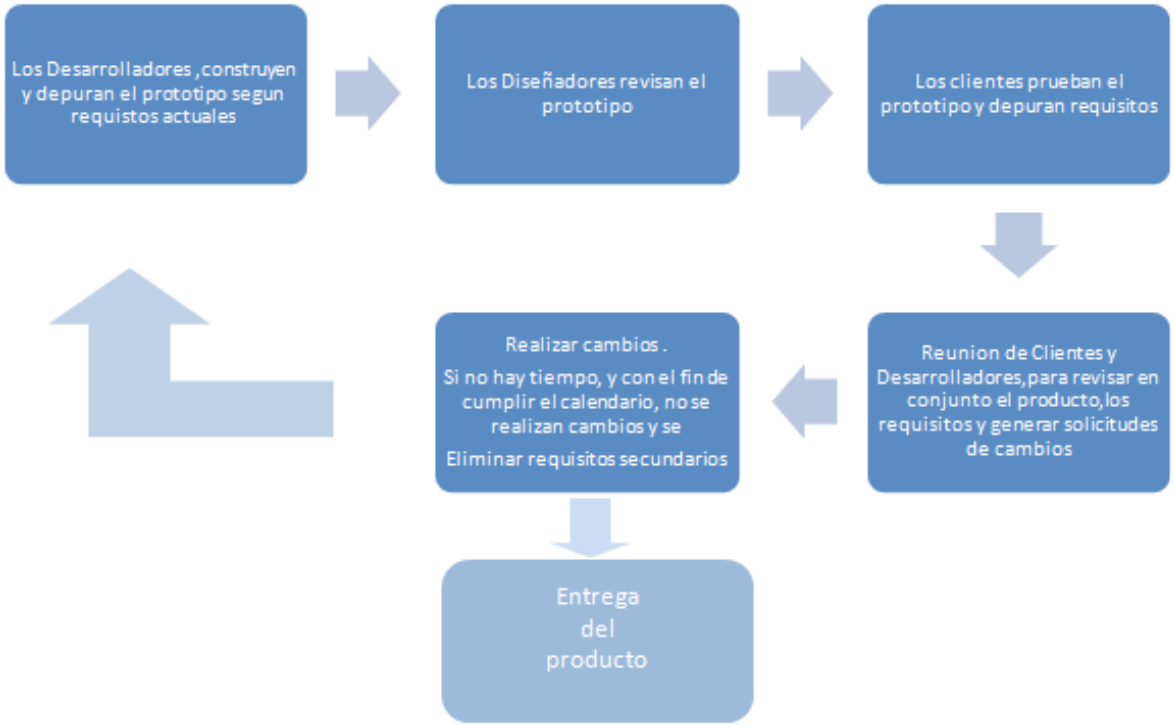


Imagen 10
Fuente: Propia.

Extreme Programming (XP)

Centrada en resaltar las relaciones interpersonales como base fundamental para el trabajo de desarrollo.

Promueve el trabajo en equipo.

Apoya el aprendizaje de los integrantes del equipo de desarrollo.

Busca crear un buen clima de trabajo.

La metodología XP se fundamenta en los siguientes elementos:

Comunicación: preguntas y contra preguntas que permitan determinar las características del sistema finalizado.

Coraje: para exponer dudas, miedos y experiencias. Un equipo de desarrollo extremo se basa en la confianza hacia sus miembros.

Simplicidad: para mantener el software lo más sencillo posible.

Feedback: capacidad de respuesta ante cambios del proyecto, tomando las retroalimentaciones del cliente, miembros del equipo y el entorno.

Un proyecto XP sigue el siguiente Ciclo de vida:

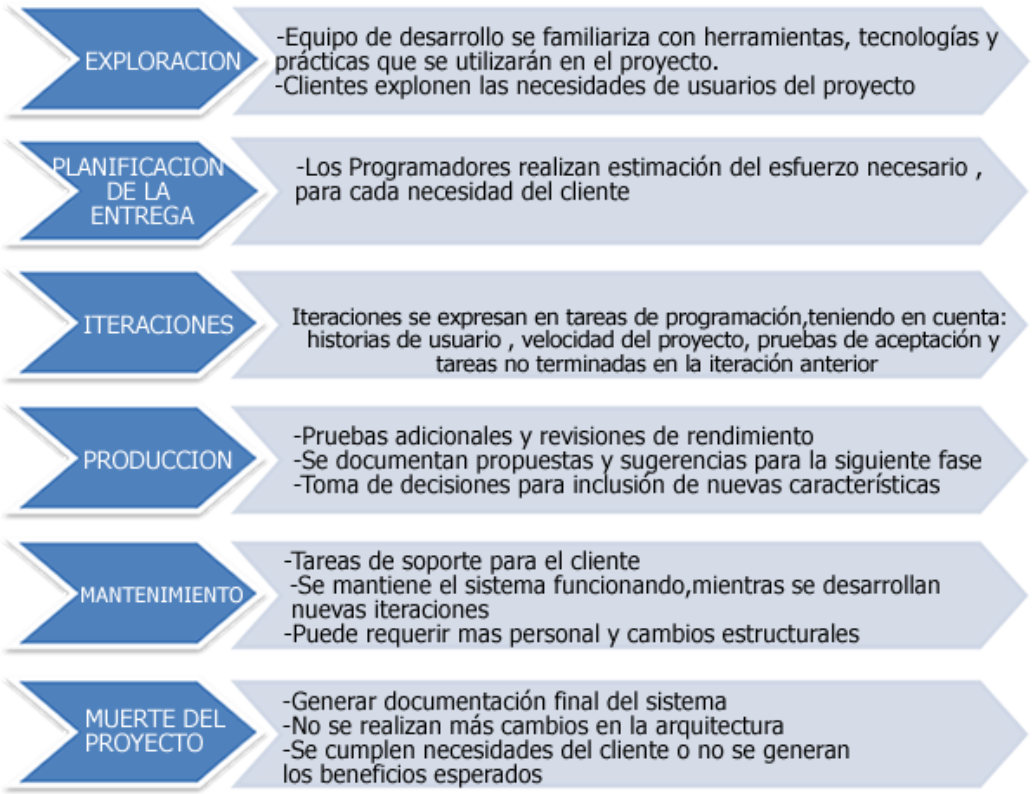


Imagen 11
Fuente: Propia.

KANBAN

- Palabra japonesa que significa "tarjetas visuales". Kan= visual, y Ban=tarjeta.
- Su objetivo es gestionar la forma en que se van completando las tareas laborales a fin de mejorar su fluidez.
- Se utiliza una tarjeta Kanban para visualizar las tareas durante el flujo de trabajo.

Principios básicos de KABAN:

- Visualizar las tareas propias de su trabajo.
- Limitar y establecer metas de trabajo en proceso.
- Realizar un seguimiento del tiempo dedicado a cada actividad.
- Lectura de indicadores visuales.
- Identificar cuellos de botella y buscar alternativas.
- Garantizar la calidad, mas no la rapidez.
- Continuo mejoramiento.
- Flexibilidad de acuerdo a necesidades.

La siguiente imagen muestra como se implementa en forma practica la metodología Kan-Ban:

Imagen 12
Fuente: http://www.snl19.es/wp-content/uploads/Tablero-kanban-izenius.jpg.png

A continuación se presentan los pasos a seguir para crear una estrategia KanBan.

1. Definir el flujo de trabajo de los proyectos:

   Cada columna corresponde a un estado de la tarea de acuerdo al proyecto, por ejemplo: Sin iniciar, en espera, en desarrollo, en pruebas diagnóstico, definición, programación, ejecución, finalizado, etc.

2. Visualizar las fases del ciclo de producción:

   - Dividir el trabajo en distintas partes como en el desarrollo incremental.
   - Cada parte se escribe en un post-it y se pega en la fase que corresponda.
   - Se busca que queden totalmente claras las tareas asignadas a cada persona del equipo de trabajo, las prioridades y metas asignadas.

3. "Stop Starting, start finishing"
    - Este es el lema de KanBan.
    - Priorizar el trabajo en curso en lugar de iniciar nuevas tareas.
    - Deben existir un número máximo de tareas en curso y para cada fase.
    - No se puede abrir otra tarea sin finalizar alguna que se encuentre en curso.
4. Control de flujo
    - Para hacer seguimiento al trabajo realizado.
    - Mantener al equipo con un flujo de trabajo constante.
    - Mantener las Tareas importantes en cola para su desarrollo.
    - Seguimiento pasivo para evitar la interrupción al trabajo que está desarrollando cada integrante.

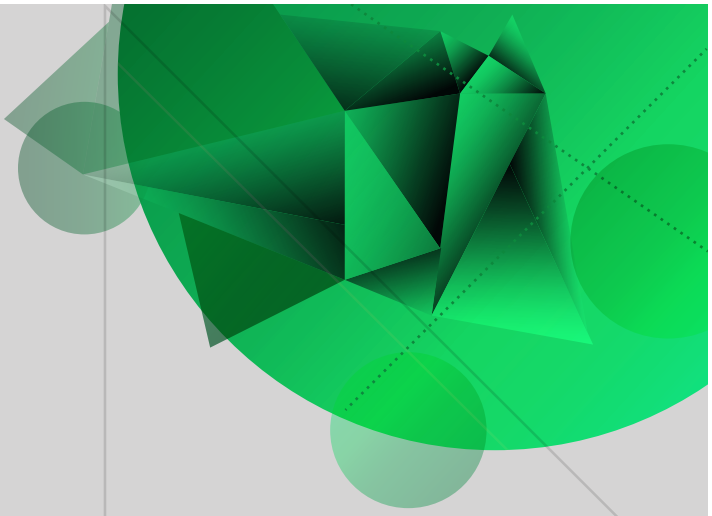
Las anteriores son algunas de las metodologías agiles que existen, la invitación es a investigar otras metodologías como:

- DSDM
- MODELADOAGIL
- LEANDEVELOP
- CRYSTALMETHOD

3

Unidad 3

Modelado de procesos



Ingeniería de software

Autor: Fredy Alonso León Socha

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

El Ingeniero de sistemas debe conocer los modelos sobre los cuales se implementan las redes de comunicación, así como también las normas que rigen estas implementaciones y componentes de hardware asociados.

En esta semana se abordarán los temas mencionados, para que de esta manera el estudiante pueda tener la fundamentación necesaria que le permita afrontar situaciones de diseño de redes e implementación del cableado estructurado que se requiere para las mismas.

U3 Metodología

El módulo se basa en la lectura individual de la presente cartilla y posterior desarrollo de actividades planteadas en la plataforma online.

Se espera una participación y acceso a plataforma a su consideración, como mejor pueda organizarse, para cubrir los objetivos de aprendizaje y participación. Por mi experiencia, recomiendo la asistencia diaria para aprovechar al máximo el curso. Es interesante seguir los hilos de participación 2-3 veces al día y dedicar uno momento para analizar y participar inteligentemente.

La comunicación conmigo la pueden realizar abiertamente a través del Foro y personalmente a través del email.

Al final de cada sección enviaré un breve mensaje resumen destacando los mejores en sus intervenciones individuales y en los trabajos de grupo de esa sección.

Aquellos que hayan llamado mi atención negativamente por su falta de participación o por lo inadecuado de las mismas les enviaré un mensaje privado.

U3

Desarrollo temático

Modelado de procesos

Conceptos de modelado

Modelo: representación de un sistema y las tareas que este realiza, de forma tal que sea lo más aproximadamente posible con la realidad del mismo. Cuando se modela un proceso mediante una representación grafica, se denomina "diagrama de proceso" y de esta forma se puede diferenciar claramente las interrelaciones, actividades, subprocesos y problemas que podrían existir.

Diagramado: tomar los procesos y subprocesos y establecerles una representación visual, de tal forma que se pueda conocer sus características tales como, tamaño, tiempos de ejecución, actividades, etc.

Dominio: contexto en el cual se genera y desenvuelve un problema.

El Modelo UML

- UML - Unified Modeling Language, es un lenguaje grafico para hacer modelos independientemente de los métodos de análisis y diseño.
- Permite representar, visualizar, especificar, construir y documentar los elementos de un sistema de software, como el diseño, comportamiento, arquitectura, etc.

UML maneja el concepto de vistas, entendida como un conjunto de diagramas que juntos muestran las funcionalidades del sistema. Las vistas que maneja UML son:

- Vista use-case: muestra las funcionalidades desde el punto de vista de actores externos.
- Vista lógica. muestra el diseño de funcionalidad interna del sistema, teniendo en cuenta su estructura estática y conducta dinámica.
- Vista de componentes: muestra cómo están organizados los componentes de código.
- Vista concurrente: muestra los aspectos de comunicación y sincronización.
- Vista de distribución: muestra la arquitectura física del sistema.

Importancia del modelado UML

Facilita la intercomunicación entre analistas, diseñadores, especialistas y programadores del proyecto; ya que UML posee más características visuales que de programación, de tal forma que todos pueden participar en el diseño y análisis del sistema.

El diseño y análisis se realiza de forma independiente al lenguaje de programación que se utilice para desarrollo del sistema y se puede establecer los requerimientos y estructuras, antes de los procesos de codificación.

Al utilizar como base un diseño visual del sistema, es más fácil detectar las relaciones, dependencias y problemas que se podrían generar en los procesos del mismo. Realizar cambios en la etapa de análisis es más fácil que hacerlos en etapas avanzadas del desarrollo.

Características principales

- Utiliza la notación orientada a objetos.
- Se basa en las especificaciones BOOCH, RUMBAUGH y COAD-YOURDON.
- Cada proyecto es dividido en igual número de diagrama, los cuales que representan las diferentes vistas y en conjunto permiten hacer una representación de la arquitectura general del proyecto.
- Utiliza diferentes niveles de abstracción para hacer la descripción de un sistema, de tal forma que el equipo de trabajo puede comprender de forma clara y detallada las características de dicho sistema.
- No es un modelo estándar para el desarrollo, sino un lenguaje de modelado.
- Los procesos de desarrollo se especifican de acuerdo al dominio y contexto del proyecto de software.
- Desarrollado para representar, visualizar, especificar, construir y documentar los elementos de un sistema de software.

Ventajas e inconvenientes

Ventajas	Desventajas
• Se puede utilizar para diferentes tipos de sistemas. • Al utilizar modelado visual, facilita su comprensión. • Útil en el modelado visual de cualquier proyecto, independientemente si es de software o no. • Promueve la reutilización. • Mejora los tiempos de desarrollo, al poder clarificar cada tarea de los actores que intervienen en el sistema. • Permite el escalamiento en sistemas complejos. • Mejora la planeación y control del proyecto de software. • Hace énfasis en la reutilización y minimización de costos.	• Limitado al modelamiento y no puede usarse como un método de desarrollo. • Se requiere adicionalmente de una metodología orientada a objetos. • Hace uso de notaciones y conceptos de la metodología orientada a objetos, por lo que quien no ha tenido experiencia previa no le será fácil aprenderlo. • No funciona en el diseño de sistemas distribuidos. • En la documentación, hacen falta muchos más ejemplos que sirvan de guía para el usuario. • Tiene fallos respecto a las necesidades de especificación del proyecto. • No se define la documentación ni el diseño de interfaces de usuario.

Cuadro 1
Fuente: Propia.

Objetivos de UML

- Ser independiente del lenguaje de programación utilizado en el proceso de desarrollo.
- Brindar la información de notaciones y semánticas que permitan abarcar el mayor numero de aspectos del modelado orientado a objetos, teniendo hacia donde apunta el desarrollo de software.
- Brindar extensiones que permitan a los proyectos implementar el meta-modelo a un bajo costo y desarrollar futuras formas de modelado utilizando como ase UML.
- Simplificar al máximo la forma de uso, manteniendo las capacidades que permitan hacer el modelamiento de diferentes tipos de sistemas.
- Convertir el lenguaje UML en un lenguaje universal y en un estándar mundial.
- Proporcionar las herramientas suficientes que permitan migrar las interfaces a bibliotecas o API, con el fin de utilizarlas en procesos de comparación y almacenamiento de componentes del modelo.

Modelo UML de un sistema

En UML se utiliza el Modelo Estático, el cual representa las clases, objetos y relaciones de un sistema, mediante "diagrama de clases" y "diagrama de objetos.

Diagrama de clases. Para definir una clase se siguen las siguientes reglas:

- La nomenclatura para niveles de acceso de métodos y atributos es la siguiente:

  + (Publico)

  # (Protegido)

  - (Privado)

- Los atributos y métodos se definen utilizando la siguiente sintaxis: *Nombre_Atributo / Método: Tipo_Dato.*
- Los parámetros se definen en forma similar: Nombre parámetro: Tipo de dato.

A continuación se muestra como se implementa una clase teniendo en cuenta las reglas antes mencionadas:

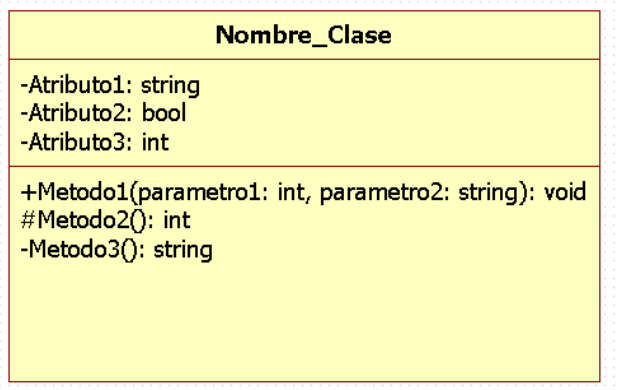


Imagen 1
Fuente: Propia.

Relaciones binarias

- Se establecen entre clases y objetos.
- Se representan mediante una línea recta indicando el nombre de la relación.
- En los extremos de la línea se indica alguno de los siguientes tipos de multiplicidad:
  - Uno a muchos (1 - *).
  - Uno a Uno (1 – 1).
  - Uno a Cero o Uno (1 – 0..1).
  - Uno a Cero o Muchos (1 – 0..*).
  - Uno a Uno o Muchos (1 – 1..*).
  - Muchos a Muchos (* - *).

Teniendo en cuenta las nomas anteriores, la implementación de las relaciones binarias se indica a continuación:

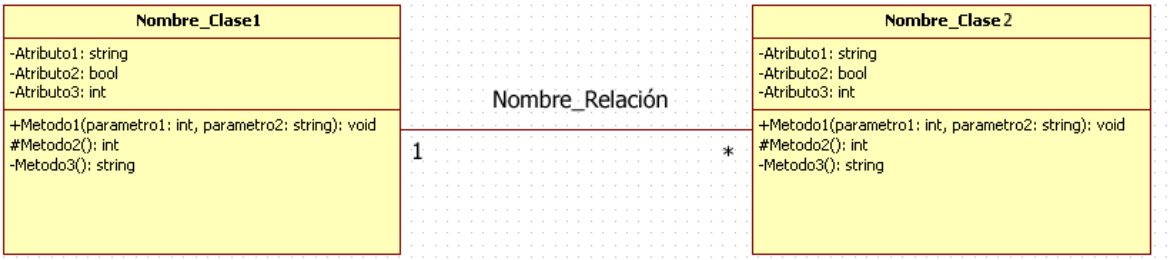


Imagen 2
Fuente: Propia.

Los otros tipos de relaciones se indican en el numeral 1.5 Relaciones

La siguiente imagen muestra un diagrama de clases para una biblioteca:

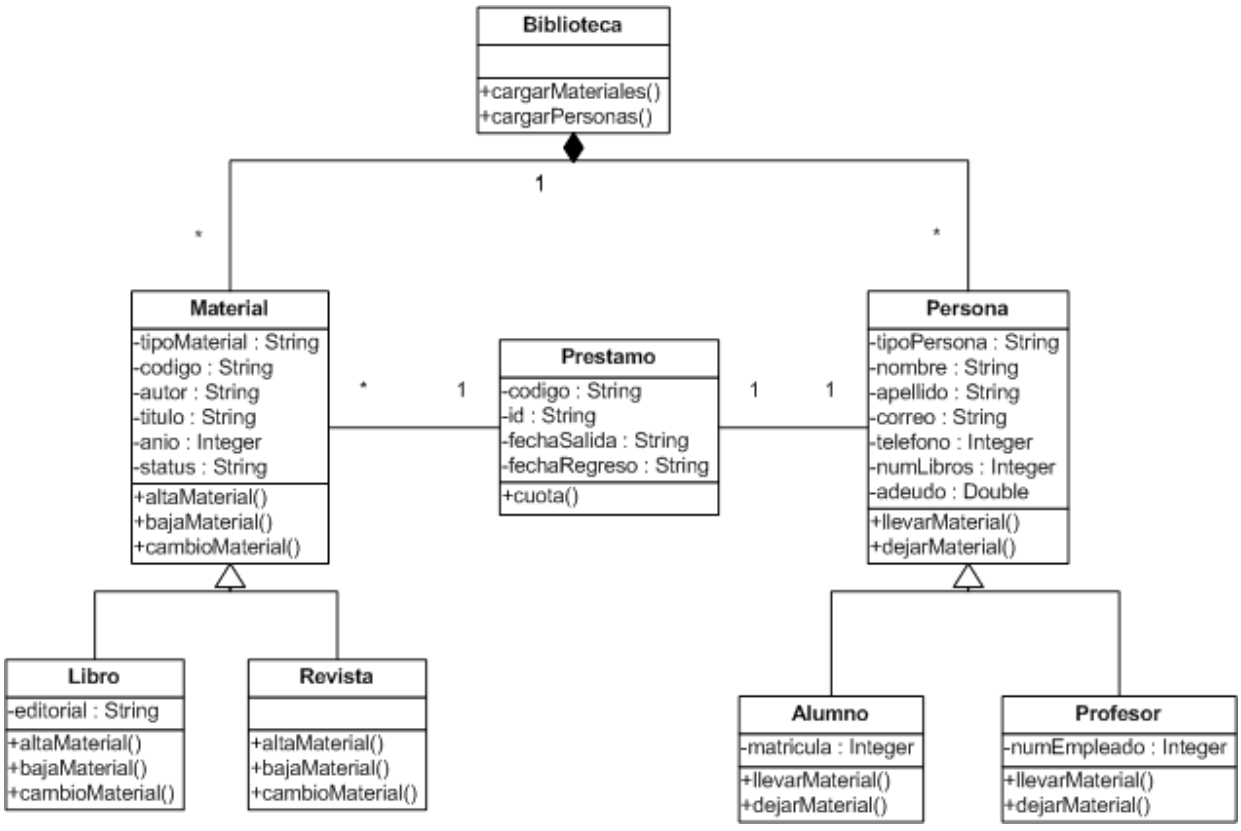


Imagen 3
Fuente: http://goo.gl/IXc1Ff

Diagrama de objetos

- Es una instancia de un diagrama de clases.
- Describen la estructura estática de un sistema.
- Incluye los objetos y los valores de sus datos.
- Se usan para probar la precisión de los diagramas de clases
- Ayudan a explicar clases y herencia
- Un objeto se define de la siguiente forma:

Nombre_Objeto: Nombre_clase	: Nombre_clase
Atributo1 = valor1 Atributo2 =valor2 Atributo3 = valor3	Atributo1: valor1 Atributo2: valor2 Atributo3: valor3

Cuadro 2
Fuente: Propia.

- Los atributos siempre llevaran un valor asignado.
- El nombre y clase siempre van subrayados.

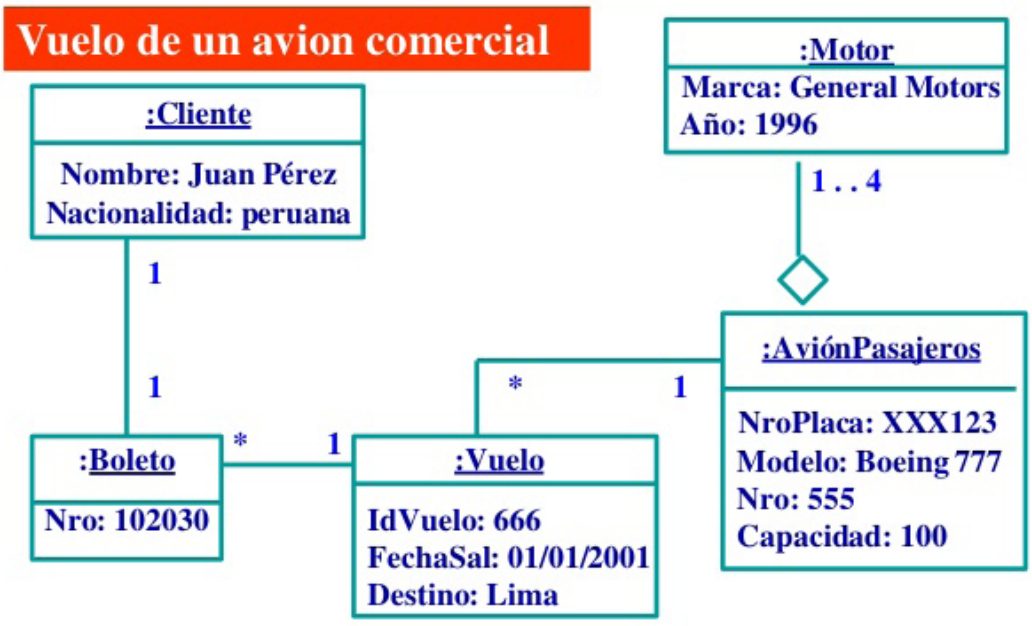


Imagen 3
Fuente: http://goo.gl/TdnSd5

Para revisar el proceso de modelamiento de un sistema, se recomienda visitar los siguientes links:

https://msdn.microsoft.com/es-es/library/bb972214.aspx

http://es.slideshare.net/RafaelDiaz12/modelado-uml-de-sistema-punto-venta

Bloques de construcción

La siguiente tabla resume los elementos que componen las 3 áreas de los bloques de construcción usados en UML:

Bloques de construcción		
Elementos	Relaciones	Diagramas
Estructurales Comportamiento Agrupación Anotación	Dependencia Asociación Generalización Realización	Clases Objetos Casos de uso Secuencia Colaboración Estados Actividades Componentes Despliegue

Cuadro 3
Fuente: Propia.

A continuación se hace una descripción de cada uno de los elementos relacionados.

Elementos estructurales

Parte estática de los modelos UML y representan elementos conceptuales o materiales. Se compone de 7 elementos:

Clases. Descripción de varios objetos que tienen los mismos atributos, operaciones, relaciones y semántica. Su representación grafica es la siguiente:

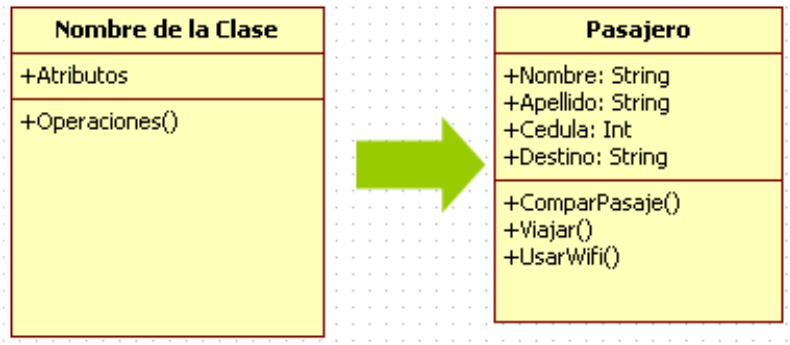


Imagen 4
Fuente: Propia.

Interfaces. Colección de operaciones y sus especificaciones, relacionadas a un servicio de una clase. Su representación grafica es la siguiente:

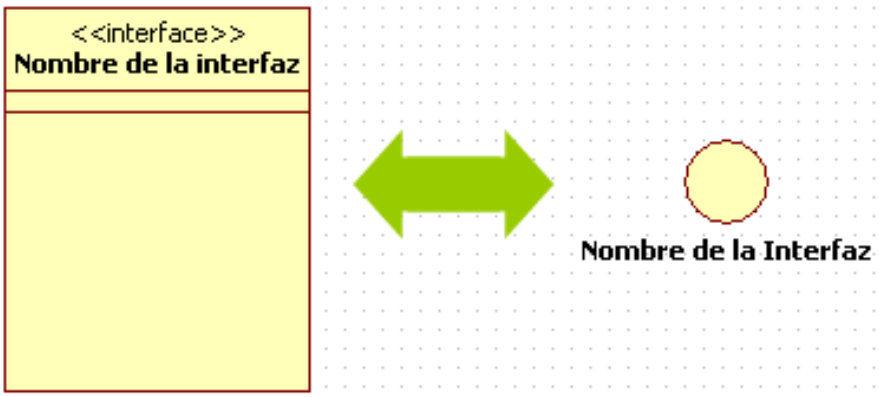


Imagen 5
Fuente: Propia.

Colaboraciones. Definen las interacciones de roles y otros elementos de las clases y en conjunto generan un comportamiento mayor. Una clase puede estar en varias colaboraciones.

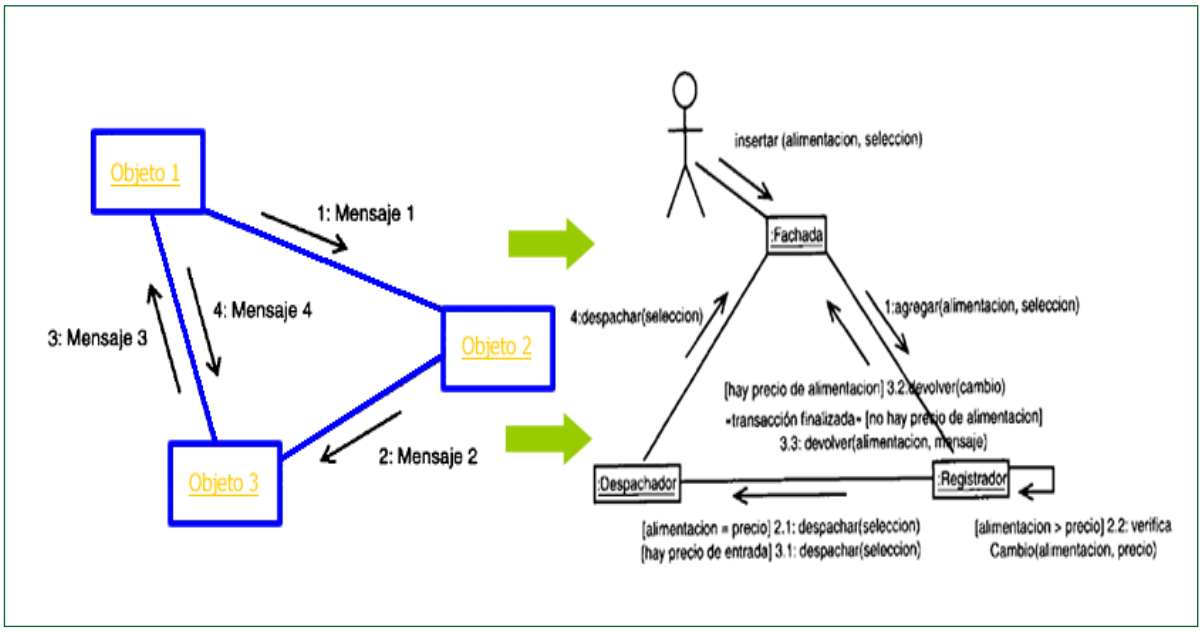


Imagen 6
Fuente: Propia, basado en http://goo.gl/TpKc9K

Casos de uso. Descripción de las diferentes secuencias que un actor realiza dentro de un sistema, las cuales generan un resultado observable y de interés para el actor. Los casos de uso se realizan por una Colaboración y se representan gráficamente por una elipse.

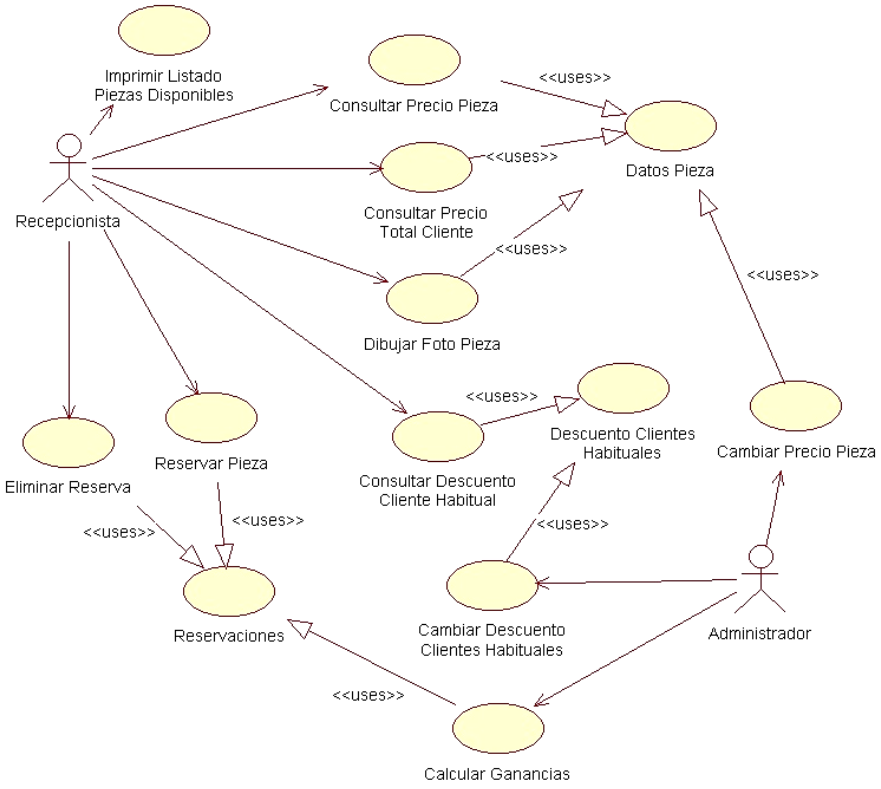


Imagen 7
Fuente: http://goo.gl/op5lCw

Clases activas. Clase en la cual los objetos tienen uno o más Hilos de ejecución, que pueden iniciar actividades de control con otros elementos de otros objetos. Gráficamente es la misma que una clase, pero con líneas más gruesas.

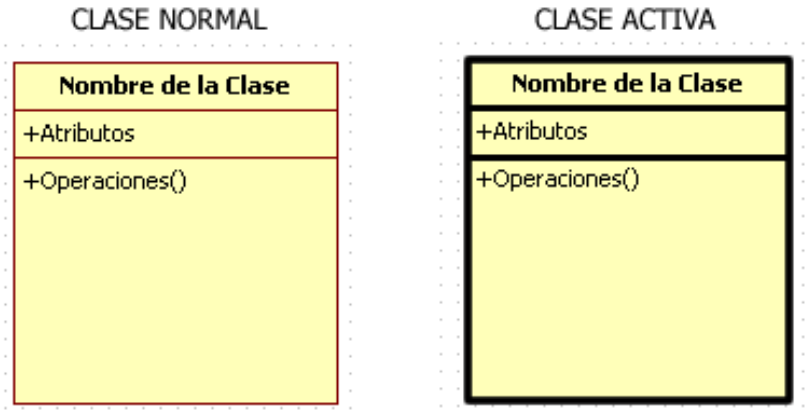


Imagen 8
Fuente: Propia.

Componentes. Representa el empaquetamiento físico de clases, interfaces, códigos fuentes y colaboraciones, como por ejemplo; COM+ o JavaBeans. Su representación física es la siguiente:

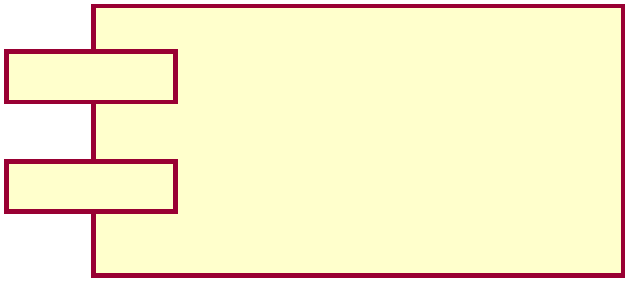


Imagen 9
Fuente: Propia.

Nodos. Representa un recurso computacional que tiene capacidad de procesamiento de información. Su representación grafica es un cubo.

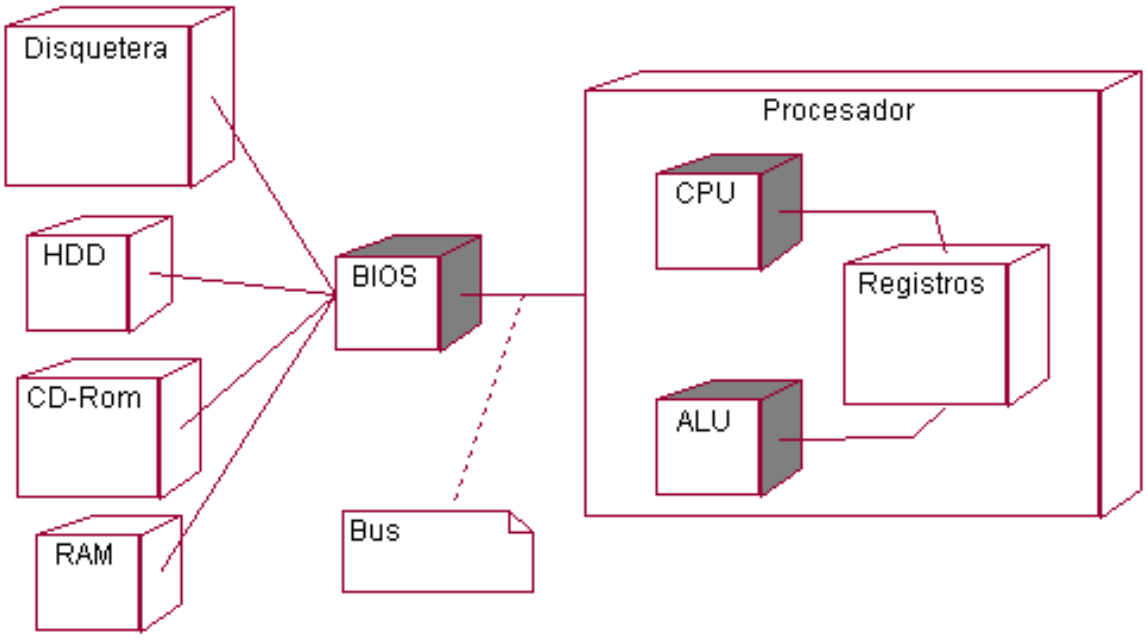


Imagen 10
Fuente: http://www.monografias.com/trabajos28/proyecto-uml/Image305.gif

Elementos de comportamiento

Interacciones	Son los diferentes mensajes, secuencias de acción y enlaces, que intercambian los diferentes objetos de un sistema, con el objetivo de cumplir una tarea específica.	Nombre de la Interacción →
Maquina de estados	Representa los diferentes estados por que pasa un objeto o una interacción ,como respuesta a un evento.	Estado

Cuadro 4
Fuente: Propia.

A continuación se muestra un diagrama que muestra los diferentes estados de un proceso y sus interacciones.

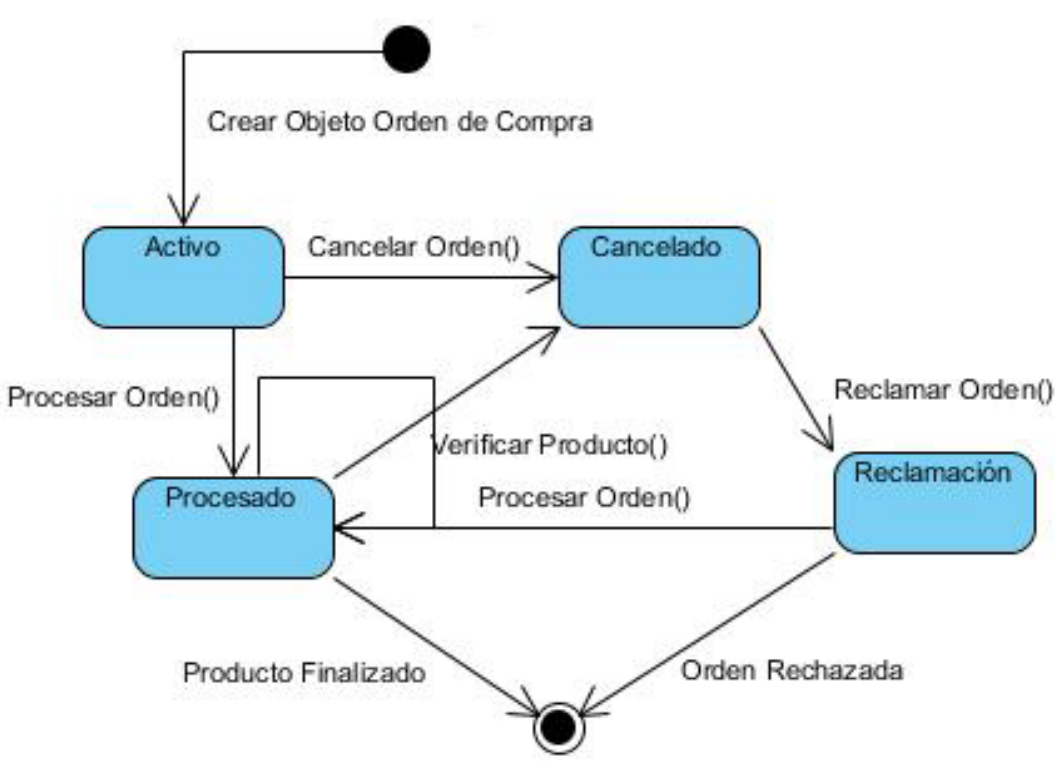


Imagen 11
Fuente: http://goo.gl/0Tn7CJ

Elementos de agrupación

Paquete. Permite organizar en grupos, los elementos estructurales, de comportamiento u otros paquetes. Su representacion grafica es la siguiente:

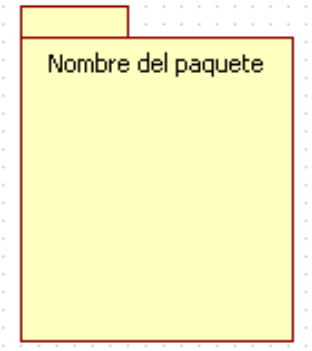


Imagen 12
Fuente: Propia.

Elementos de anotación

Notas. Se utiliza para mostrar restricciones o comentarios de un elemento o colección de elementos. Su representación grafica es la siguiente:

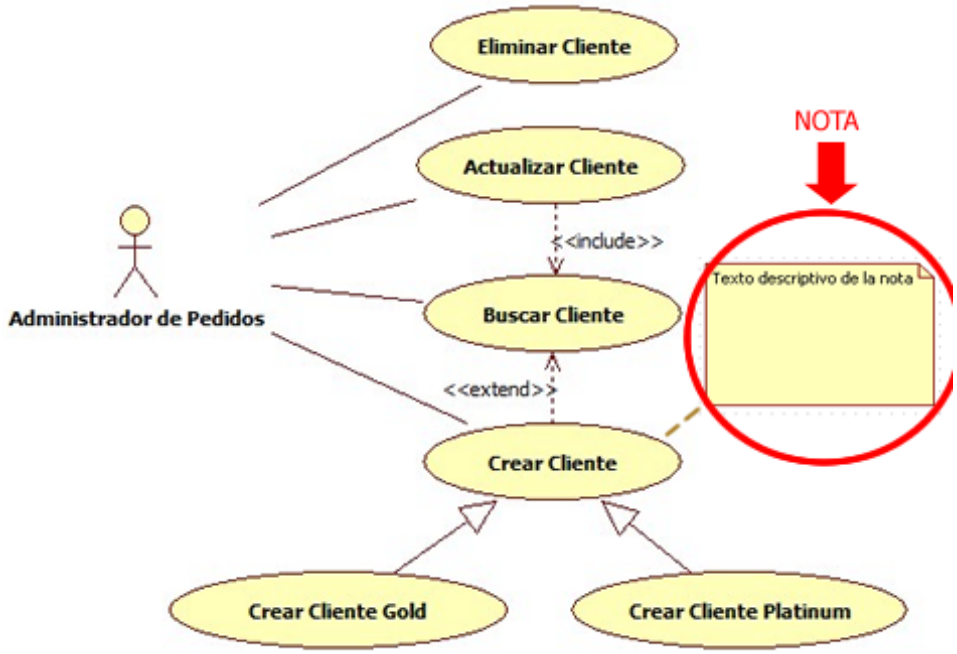


Imagen 13
Fuente: Propia.

Relaciones

Tipo	Descripción	Representación
Dependencia	Relación entre dos elementos en la cual cualquier cambio en el elemento independiente puede afectar al elemento dependiente.	
Asociación	Relación de tipo estructural, que describe las conexiones entre objetos. Un tipo especial de asociación es la *agregación*, que representa la relación entre un todo y sus partes.	Asociación Agregación
Generalización	Los objetos de un elemento especializado (hijo), pueden sustituir a los objetos del elemento general (padre).Tanto hijo como padre, comparten la estructura y comportamiento.	
Realización	Acciones entre Interfaces/Clases y componentes que las ejecutan o entre. casos de uso y las colaboraciones que las ejecutan.	

Cuadro 5
Fuente: Propia.

Diagramas estructurales y de comportamiento

Área	Vista	Diagramas	Conceptos	Uso
Estática	Vista lógica	De clases	Clase, asociación, generalización, interfaz dependencia, realización.	Modelar la estructura estática de las clases.
		De casos de uso	Casos de uso, actor, asociación, extensión, generalización.	Modelar los procesos de negocio.
	Vista física	De componentes	Componente, interfaz, dependencia, realización.	Modelar componentes del sistema.
		De implementación	Nodo, componente, dependencia, realización.	Modelar la distribución del sistema.
Dinamica	Vista lógica	De estados	Estado, evento, transición, acción.	Modelar el comportamiento de los objetos.
		De actividad	Estado, actividad, transición, determinación, división, unión.	Modelar el comportamiento de los casos de uso, objetos u operaciones.
	Vista física	De secuencia	Interacción, objeto, mensaje, activación.	Modelar el paso de mensajes entre objetos.
		De colaboración	Colaboración, interacción, rol, mensaje.	Modelar interacciones entre objetos.

Cuadro 6
Fuente: Propia.

Mecanismos comunes

Especificaciones

Permite explicar la sintaxis y semántica cada elemento del sistema, como por ejemplo; la información de operaciones, atributos, comportamiento y signaturas.

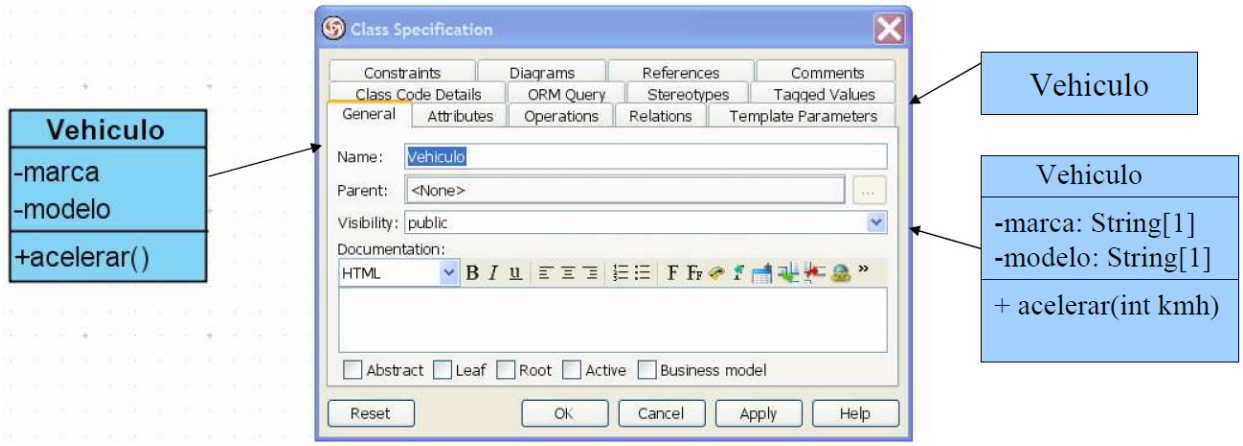


Imagen 14
Fuente: http://goo.gl/Xohpny

Adornos

Representa los detalles adicionales de un elemento, por ejemplo el tipo de clase, si los atributos y operaciones son visibles o no, tipos de datos de los atributos, etc.

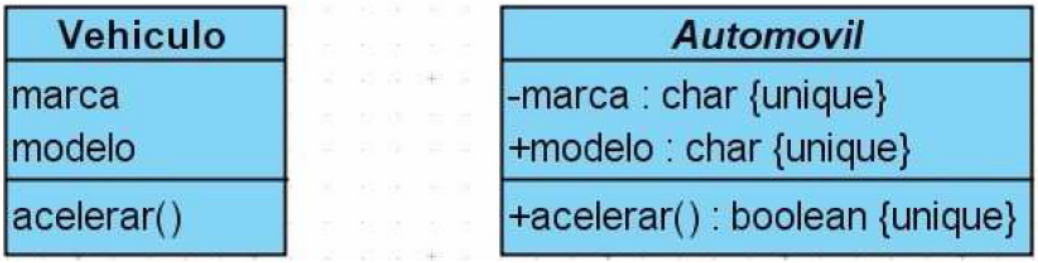


Imagen 15
Fuente: http://goo.gl/Xohpny

Divisiones comunes

Mecanismos de extensibilidad

Permiten ampliar el lenguaje UML, utilizando alguno de los siguientes perfiles:

- Estereotipos: añadir nuevos bloques de construccional.
- Valores etiquetados: modificar o caracterizar la especificación de los nuevos bloques de construcción.
- Restricciones: añadir nuevas reglas o modificar las existentes.

Reglas

Se refieren a cómo pueden combinarse los elementos, relaciones y diagramas. Estas se agrupan de la siguiente manera:

- Nombres: como identificar los elementos, relaciones y diagramas.
- Alcance: contexto que le da el significado a un nombre.
- Visibilidad: la forma en que se pueden ver y utilizar los nombres.
- Integridad: relación apropiada y consistente entre los elementos.
- Ejecución: como se llevan a cabo los procesos.

Vistas arquitecturales

El modelo 4+1 fue diseñado por Philippe Kruchten para "describir la arquitectura de sistemas software, basados en el uso de múltiples vistas concurrentes". La siguiente imagen muestra la estructura del modelo:

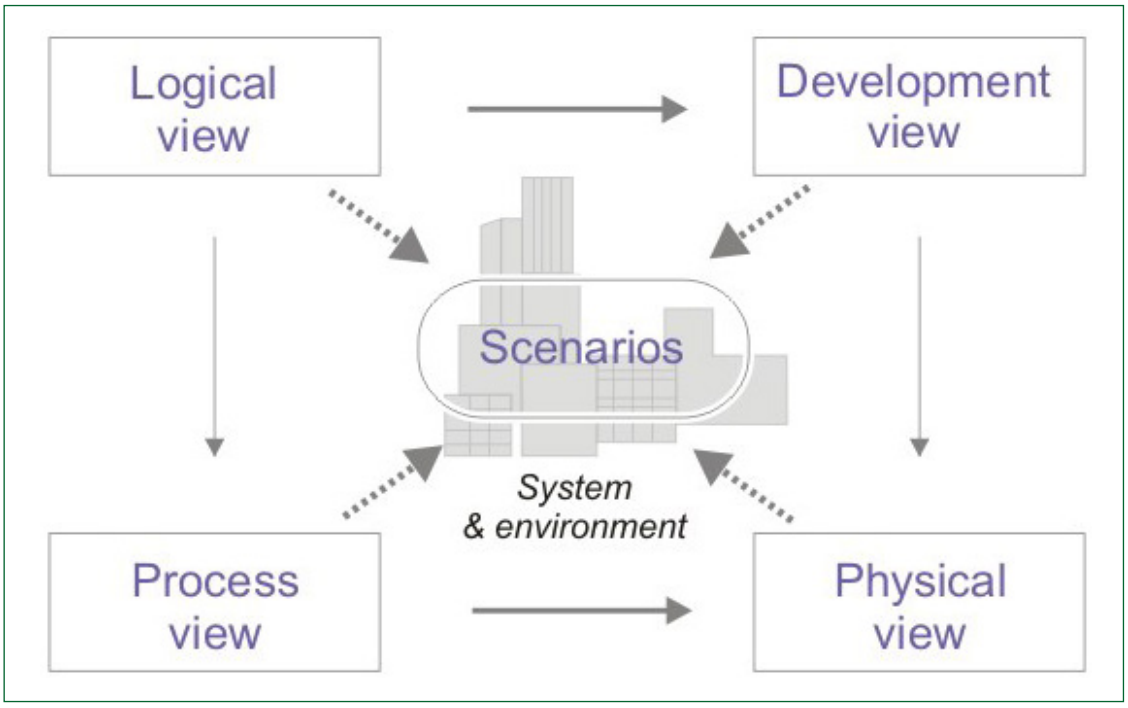


Imagen 16
Fuente: https://goo.gl/Uk2tKf

Se dividen en físicas y Lógicas, como se muestra en el siguiente diagrama:

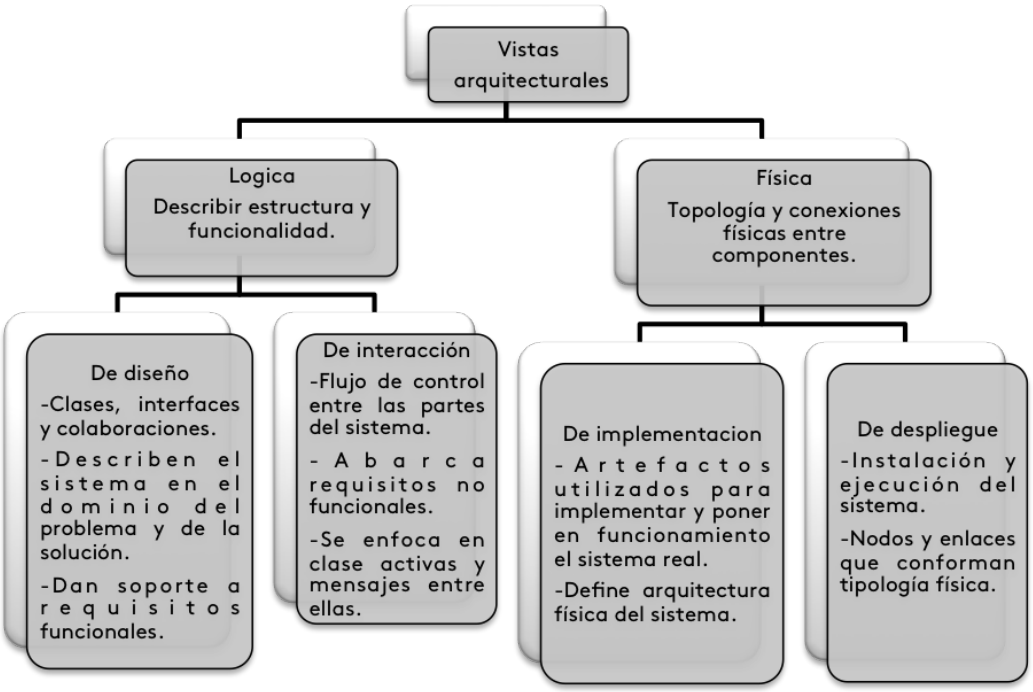
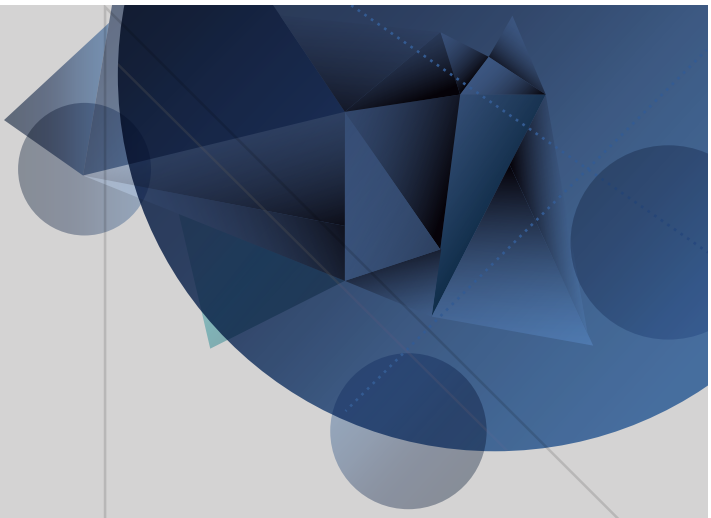


Figura 1
Fuente: Propia.

4

Unidad 4

Conceptos de seguridad



Ingeniería de software

Autor: Fredy Alonso León Socha

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

Es importante conocer las líneas de equipos y terminales que fabrican las diferentes empresas desarrolladoras de hardware en telecomunicaciones, a fin de tener un concepto global al momento de realizar una implementación de un sistema de comunicaciones, hardware, etc.

Esta cartilla presenta un resumen de los equipos fabricados las marcas Intel, AMD, Motorola e IBM.El documento se enfoca por un lado a presentar una breve descripción de cada empresa y relacionar principalmente las últimas líneas de equipos fabricados en las áreas computación, telefonía móvil o comunicaciones, en base a cada empresa.

U4 Metodología

El módulo se basa en la lectura individual de la presente cartilla y posterior desarrollo de actividades planteadas en la plataforma online.

Se espera una participación y acceso a plataforma a su consideración, como mejor pueda organizarse, para cubrir los objetivos de aprendizaje y participación. Por mi experiencia, recomiendo la asistencia diaria para aprovechar al máximo el curso.

La comunicación conmigo la pueden realizar abiertamente a través del Foro y personalmente a través del email.

Se realizará una teleconferencia semanal de un tema especifico, en el cual se presentarán ejemplos para ampliar la temática semanal.

En cada semana se publica el material de estudio que apoyara el desarrollo de los contenidos temáticos correspondientes a dicha semana, ya sea en recursos bibliográficos, archivos digitales o referencias electrónicas (páginas WEB).

U4

Desarrollo temático

Conceptos de seguridad

Bien informático	Elementos que componen un sistema informático.
Sistema informático	Conjunto de bienes informáticos que ejecutan unas tareas específicas.
Amenaza	Posible riesgo que puede causar daño a un bien informático.
Riesgo	Probabilidad que una amenaza se materialice.
Análisis de riesgo	Proceso que permite determinar vulnerabilidades de un sistema, las posibles amenazas y su probabilidad de ocurrencia e impacto.
Riesgo residual	Riesgo después de haber implementado controles de seguridad para minimizarlo.
Impacto	Daño producido o causado por una amenaza.
Seguridad	Minimizar los riesgos de un bien informático a niveles adecuados.
Vulnerabilidad	Debilidades o aspectos atacables de un sistema informático y que permiten calificar el nivel de riesgo.

Cuadro 1
Fuente: Propia.

Propiedades del software seguro

Propiedades fundamentales

Confidencialidad: debe asegurar que sus características, contenidos, datos solo puedan ser accedidos por para usuarios autorizados.

Integridad: debe ser resistente y flexible a modificaciones no autorizadas del código, configuraciones, ajustes por parte de usuarios autorizados.

Disponibilidad: debe estar con total funcionalidad y accesibilidad para usuarios autorizados en el momento que lo requieran y que puedan realizar en forma corecta las tareas para las cuales fue desarrollado:

Responsabilización (accountability): registro y trazabilidad de acciones relacionadas con la seguridad del software, que debe desarrollar una entidad usuario, a fin de establecer responsabilidades y seguimientos.

No repudio. habilidad para evitar que las entidades usuario de un software, no asuman la responsabilidad de acciones que han sido ejecutadas sobre el mismo, de tal forma que no se pueda eludir la propiedad responsabilización (accountability).

Propiedades conducentes

Dependability: asegura que el software siempre funcionara sin defectos y debilidades

Correcto: operar incluso en aquellas condiciones en las que se presenta un ataque. Deben especificarse requerimientos de comportamiento seguro, para garantizar que un software correcto sea seguro.

Predecible: el software siempre ejecutara las operaciones implementadas bajos las condiciones de ambiente, entradas sobre las cuales fue diseñado.

Confiable: la ejecución debe ser predecible y correcta, aun cuando existan defectos no intencionales, debilidades o cambios en el ambiente en el cual se ejecuta.

Protección: si el sistema sufre alguna falla o ataque debe funcionar parcialmente o en modo seguro para evitar pérdidas humanas, de activos, medio ambiente, etc.

Metodologías vigentes

Correctness by Construction (CbyC)

Conceptos

Para cuando se requiere de niveles críticos de seguridad y que además se puedan demostrar. Busca minimizar el nivel de defectos y una alta resistencia al cambio. Esto se logra bajo los siguientes principios:

Que los errores sean difíciles de introducir en el software.

Que si dichos errores se han inyectado, estos deben removerse los más pronto posible.

Generar pruebas de aptitud como un subproducto del proceso y de esta forma usarlas en todo el desarrollo.

Lo anterior se logra a través del desarrollo de las siguientes actividades:

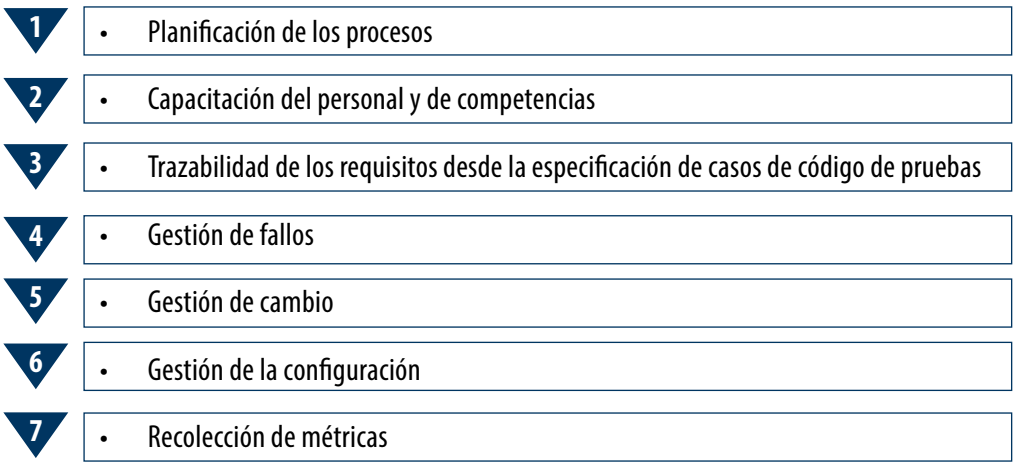


Figura 1
Fuente: Propia.

Para el desarrollo de las actividades planteadas, se utilizan las técnicas que se relacionan a continuación:

- Se hacen notas de audio para disminuir ambigüedades en la especificación de requisitos.
- Se hace una estricta validación de los productos que se entregan en cada etapa de desarrollo.
- Se utiliza el desarrollo incremental. Cada código escrito se debe ir verificando.
- Se evita al máximo la repetición de requisitos ambiguos.
- Simplificar al máximo las cosas, para que sea fácil su revisión y desarrollo.
- Hacer una gestión adecuada de los riesgos para poder tener una mejor respuesta ante las situaciones.
- Tener un pensamiento fuerte enfocado a lograr los objetivos.

Fases de CbyC

La siguiente figura muestra el proceso de desarrollo que en sigue en CbyC:

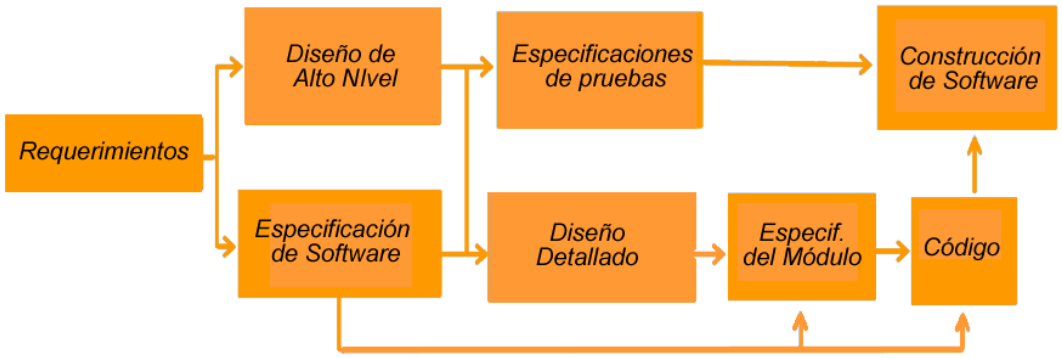


Imagen 1
Fuente: Propia.

Fase de requerimientos

- Especificar el propósito, funciones, requerimientos no funcionales, requerimientos de usuario con diagramas contextuales, de clase y definiciones operativas.
- Pasar los requerimientos por un proceso de trazabilidad y requerimiento de seguridad a la amenaza correspondiente.

Fase de diseño de alto nivel

- Definir estructura interna del sistema: funcionalidades, bases de datos, transacciones, comunicaciones, interacciones y especificar requerimientos no funcionales de protección y seguridad.

Fase de especificación del software

- Documentar especificaciones de interfaz de usuario, documentar especificaciones de niveles superiores, desarrollar un prototipo para validación.

Fase de diseño detallado

- Definir los módulos y procesos con sus funcionalidades usando notación formal para evitar ambigüedades de interpretación en el diseño.

Fase de especificación de los módulos

- Definir estado y comportamiento encapsulado de los módulos definidos en la etapa anterior, según el flujo de información, con el enfoque de bajo acoplamiento y alta cohesión.

Fase codificación

- Aplicar el lenguaje SPARK[1] en partes criticas del sistema.
- Realizar pruebas de análisis al código con el fin de eliminar errores, así como su respectiva actualización.

Fase de las especificaciones de las pruebas

- Efectuar pruebas de valores limites, de comportamiento, y de requerimientos no funcionales a nivel de sistema.
- De acuerdo a especificaciones del software, requerimientos y diseño de alto nivel.

Fase de construcción del software

- Se basa en el desarrollo ágil entregando en una primera versión, el esqueleto completo del sistema con interfaces, comunicación; funcionalidad muy limitada que aumenta con cada iteración del ciclo.

1 Lenguaje de programación diseñado para sistemas en los que se requiere alta integridad

Security Development Lifecycle (SDL)

Conceptos

- Propuesto por Microsoft en 2004.
- 16 actividades enfocadas a mejorar la seguridad del desarrollo de un producto de software.
- Se realiza un modelado de amenazas con el fin que los desarrolladores puedan encontrar código vulnerable de ataques.

Fases de la Metodología SDL

El proceso de desarrollo SDL plantea 2 opciones:

Versión rígida

- Apropiada para equipos de desarrollo y proyectos grandes, que no cambien demasiado durante el proceso.
- Frecuencia de actividades para aseguramiento de calidad es más lenta.

Orientada al desarrollo ágil

- Usa como base el desarrollo incremental.
- Frecuencia de actividades para aseguramiento de calidad es más rápida.
- Para desarrollos enfocados a la web.

En la siguiente figura se muestran las etapas que sigue el proceso de desarrollo SDL y las 16 actividades que se deben realizar.

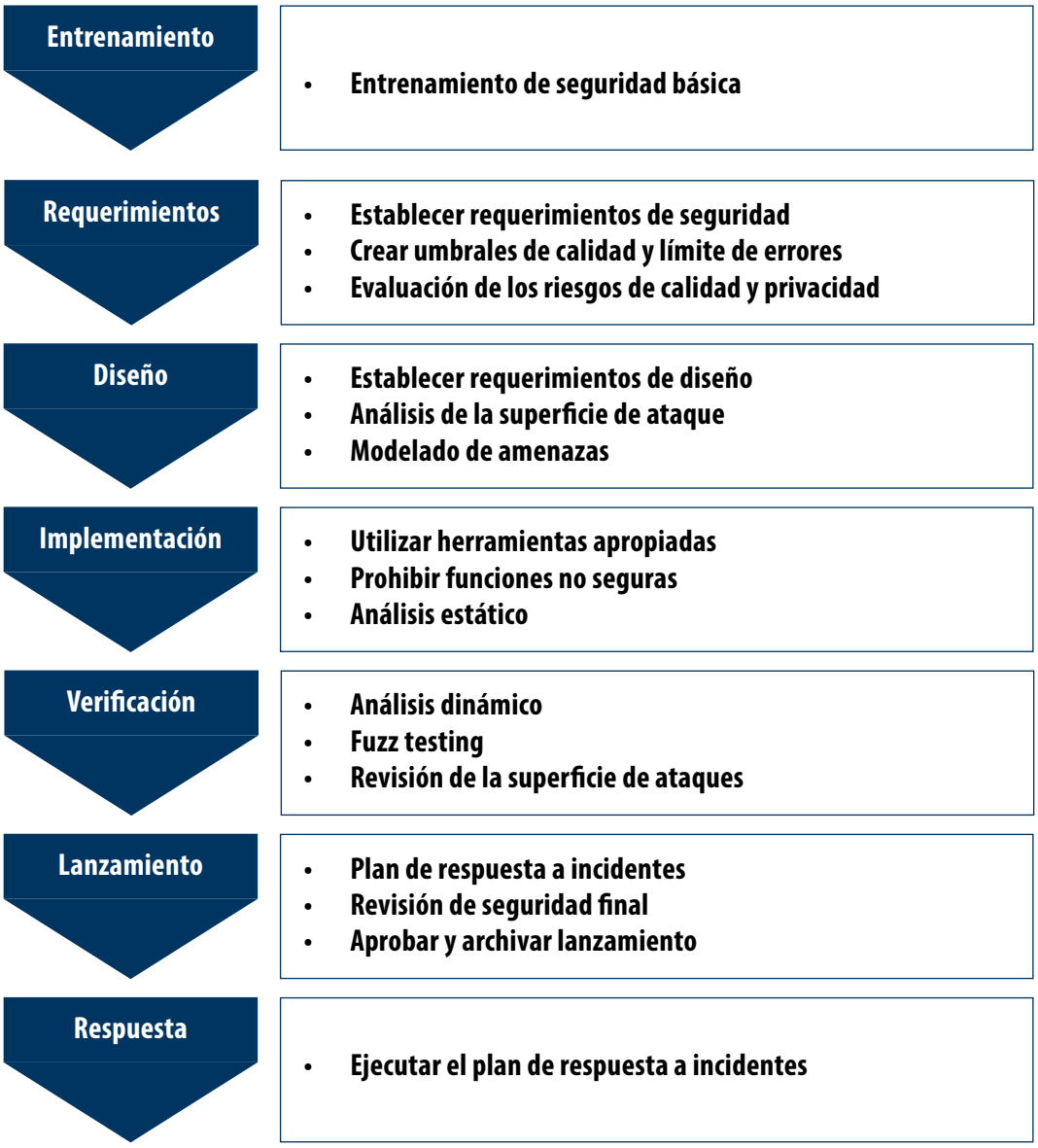


Figura 2
Fuente: Propia.

Fase de entrenamiento: el equipo de desarrollo debe recibir capacitación en conceptos básicos y tendencias en el área de seguridad y privacidad. El área técnica (desarrolladores, evaluadores y administradores de programas) debe recibir capacitación (por lo menos una vez al año) en las siguientes temáticas:

- Conceptos fundamentales de seguridad.
- Mecanismos de defensa.
- Principio de privilegios mínimos.
- Modelos de riesgos.
- Saturaciones de búfer.
- Inyección de código SQL.
- Criptografía débil.
- Evaluación de riesgos.
- Procedimientos de desarrollo de privacidad.

Fase de requerimientos: un consultor de seguridad apoya al equipo de desarrollo en revisión de planes, generando recomendaciones con el fin de cumplir metas de seguridad propuestas. Adicional a esto, el equipo de producción también debe:

- Determinar la forma en que deberá integrarse la seguridad en el proceso de desarrollo.
- Identificar los objetivos de seguridad.
- Identificar la forma que se integrará el software en conjunto.
- Verificarán que sean incluidos todos los requerimientos de seguridad.

Fase de diseño: identificar requerimientos y la estructura que llevara el software. Se identifican los requerimientos y una arquitectura segura del software, también unas guías de diseño en la cual queden identificados los componentes vulnerables. Para esto se utiliza el enfoque de privilegios mínimos y la reducción del área de ataques. Se realiza un modelado de las posibles amenazas componente por componente, estableciendo la probabilidad que ocurran y las medidas a tomar para minimizar ese riesgo.

Fase de implementación. codificar, probar e integrar el software, teniendo en cuenta el modelado de amenazas. Esto con el fin de evitar inyección de fallas, detectar vulnerabilidades. Se aplican técnicas de Fuzz Testing, consistente en inyectar al software datos al azar, inválidos y no esperados. Se utilizan herramientas de escaneo de Microsoft.

En el siguiente link podrá conocer algunas de estas herramientas:

http://www.caminogeek.com/lista-de-herramientas-de-seguridad-gratuita-de-microsoft/

Finalmente se hace una nueva revisión de código con el fin de identificar vulnerabilidades que no fueron detectadas por las herramientas de escaneo.

Fase verificación: en esta fase ya se tiene un software beta totalmente funcional. Lo que se busca es revisar exhaustivamente el código y ejecutar pruebas en aquellas partes del software que habían sido identificadas como vulnerables.

Fase de lanzamiento: se hace una revisión final para saber el nivel de seguridad del producto y nivel de probabilidad de soportar ataques, una vez que se libere al cliente. Si se encuentran vulnerabilidades, se regresa a la fase anterior hasta solucionar estas fallas.

Fase de respuesta: debido a que no es posible entregar un software 100% seguro, el equipo de desarrollo debe estar preparado para atender los incidentes de seguridad que puedan generarse y hacer una retroalimentación de los errores para proyectos futuros.

Cigital Touchpoints

Conceptos

- Es un consolidado de mejores prácticas aplicadas a seguridad.
- Son una mezcla de actividades destructivas (ofensa) y constructivas (defensa). Actividades destructivas como ataques, vulnerabilidades, y software de última hora. y las Actividades constructivas como el diseño, la defensa, y la funcionalidad.
- Fueron publicadas por primera vez en 2004 en la revista "IEEE Security & Privacy".
- Aunque no se tiene que adoptar los 7 Touchpoints para empezar a construir la seguridad de un software, es muy recomendable hacerlo.
- Están diseñados para llenar la brecha existente entre la teoría y práctica, algo que se puede hacer sólo a través de la adopción común de las mejores prácticas.

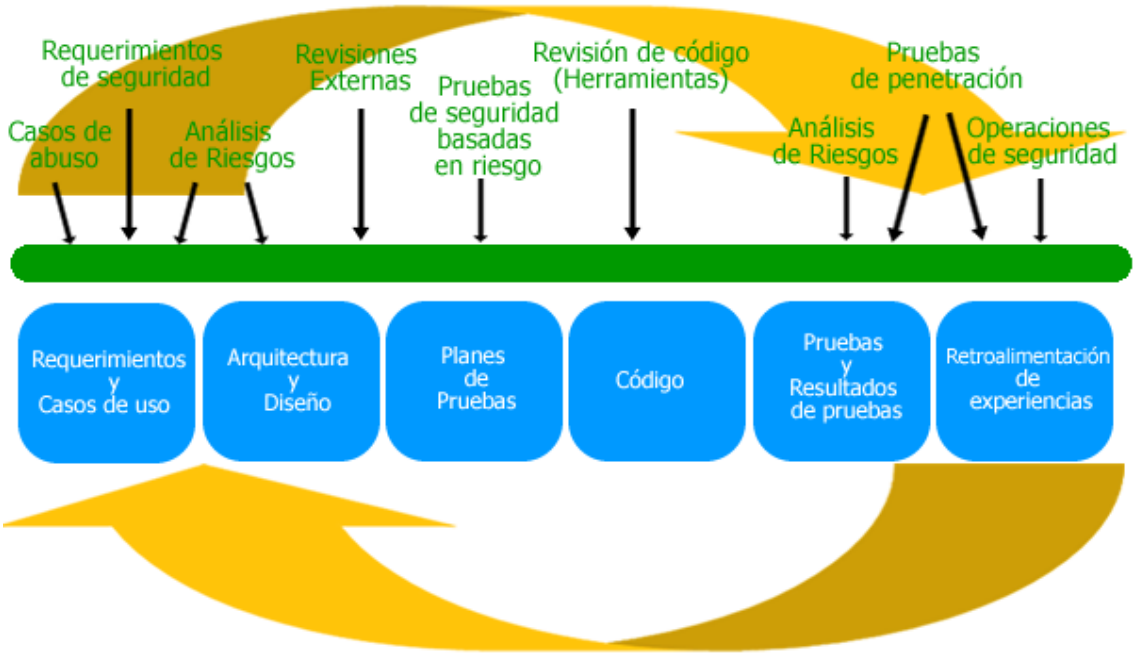


Imagen 2
Fuente: Propia, basado en: http://www.swsec.com/resources/touchpoints/

Los 7 TouchPoint

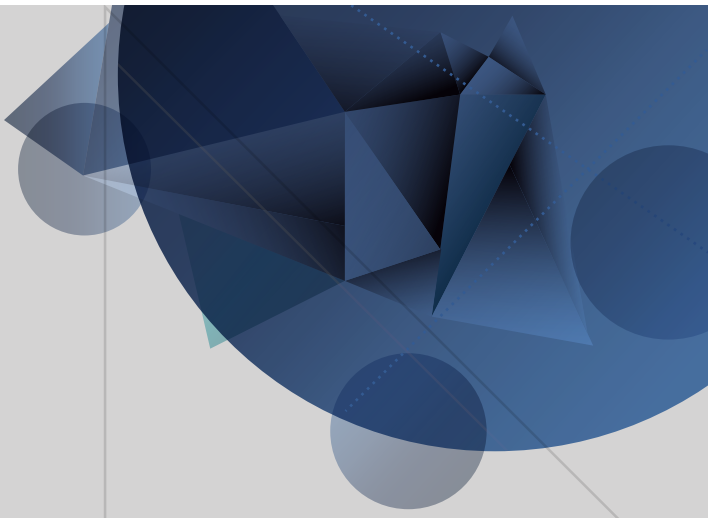
A continuación se realiza una explicación resumida de cada uno de los 7 Touchpoint:

1. Revisión de código: debe convertirse en una práctica necesaria, soportándose con la utilización de herramientas de escaneo de código. Este tipo de seguridad debe ser implementada a lo largo del ciclo de vida del desarrollo
2. Análisis de la arquitectura de riesgos: diseñadores, arquitectos y analistas deben identificar y documentar claramente los posibles ataques; en la etapa de la arquitectura basada en las especificaciones, así como en la etapa de diseño de la clase-jerarquía. Por otro lado, los analistas de seguridad descubrir y clasificar defectos de arquitectura para que pueda comenzarse una mitigación de riesgos. Es recomendable que el análisis de riesgos este presente durante todas las etapas del ciclo de vida con el fin de evitar problemas durante el camino.
3. Pruebas de penetración: proporcionan una buena comprensión del software en su entorno real, siempre y cuando dichas pruebas estén acordes con la arquitectura del software, de lo contrario no reflejara la realidad de los riesgos de seguridad.
4. Pruebas de seguridad basadas en riesgo: un buen plan de pruebas debe abarcar dos estrategias:
    - Probar la funcionalidad de seguridad con técnicas estándares funcionales de pruebas.
    - Pruebas de seguridad basado en el riesgo sobre la base de patrones de ataque.
5. Casos de abuso: describen el comportamiento del sistema en situaciones de posibles ataques. La construcción de casos de abuso requiere una cobertura explícita de lo que debe ser protegido, de quien, y por cuánto tiempo
6. Requerimientos de seguridad: la seguridad debe integrarse explícitamente en el nivel de requisitos. Unos buenos requisitos de seguridad cubren la seguridad funcional manifiesta (por ejemplo, el uso de la criptografía aplicada) y las características emergentes (Se capturan por los casos de abuso y patrones de ataque).
7. Operaciones de seguridad: el monitoreo del comportamiento del software durante su uso en ataques de seguridad, es una técnica defensiva esencial y debe ser realizada por personal de operaciones. Las experiencias adquiridas frente a estas situaciones deben retroalimentarse hacia el proceso de desarrollo del software.

4

Unidad 4

Metodologías vigentes



Ingeniería de software

Autor: Fredy Alonso León Socha

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

Es importante conocer las líneas de equipos y terminales que fabrican las diferentes empresas desarrolladoras de hardware en telecomunicaciones, a fin de tener un concepto global al momento de realizar una implementación de un sistema de comunicaciones, hardware, adquisición de software, etc.

Esta cartilla presenta un resumen de los equipos fabricados las marcas Hewlett Packard, Dell, Sun Microsystems y Apple. El documento se enfoca por un lado a presentar una descripción de cada empresa y relacionar en forma general, los productos de hardware y software generados por las mismas.

U4 Metodología

El módulo se basa en la lectura individual de la presente cartilla y posterior desarrollo de actividades planteadas en la plataforma online.

Se espera una participación y acceso a plataforma a su consideración, como mejor pueda organizarse, para cubrir los objetivos de aprendizaje y participación. Por mi experiencia, recomiendo la asistencia diaria para aprovechar al máximo el curso.

La comunicación conmigo la pueden realizar abiertamente a través del Foro y personalmente a través del email.

Se realizará una teleconferencia semanal de un tema especifico, en el cual se presentarán ejemplos para ampliar la temática semanal.

En cada semana se publica el material de estudio que apoyara el desarrollo de los contenidos temáticos correspondientes a dicha semana, ya sea en recursos bibliográficos, archivos digitales o referencias electrónicas (páginas web).

U4 Desarrollo temático

Metodologías vigentes

Common Criteria (Criterios Comunes)

Conceptos

- Se encuentran estandarizados bajo la serie de normas ISO/IEC 15408 (ISO 15408-1,2005), (ISO 15408-2,2008) y (ISO 15408-3,2008).
- Web oficial: http://www.commoncriteriaportal.org/
- Se definen un conjunto de criterios estándar usados como referencia para evaluar las propiedades y características de seguridad de productos de software o sistemas de Tecnologías de Información.
- La implementación de dichos criterios permite una equiparación entre los resultados de diferentes e independientes evaluaciones, pudiéndose obtener una certificación oficial de nivel de seguridad que puede satisfacer

Clasificación

La clasificación se establece de la siguiente manera:

Clase. Conjunto de familias que comparten un mismo objetivo de seguridad.

Familia: grupo de componentes que comparten objetivos de seguridad pero con diferente énfasis o rigor.

Componente: pequeño grupo de requisitos específicos y detallados. Es el menor elemento seleccionable para incluir en los documentos de Perfiles de Protección (PP) y Especificación de Objetivos de seguridad (ST).

Teniendo en cuenta lo anterior, a continuación se relacionan los requisitos de seguridad relacionados con la autenticación:

Clase:

- Identificación y autenticación.

Familias de la clase:

- Fallos de autenticación.
- Definición de atributos de usuario.
- Autenticación de usuario.
- Identificación de usuario.

Componentes de la familia "Autenticación de usuario"

- Tiempo de espera para la autenticación.
- Acciones antes de autenticar.
- Mecanismos de autenticación simple.
- Mecanismos de autenticación múltiple.

Fases

El proceso inicia con las siguientes actividades:

- Definir los principios, conceptos y modelo general de la evaluación de la seguridad.
- Establecer la forma en la que se realizaran las especificaciones formales de sistemas o productos de TI de acuerdo a los aspectos de seguridad de la información y su tratamiento.

En esta fase se manejan los siguientes conceptos:

Protection Profile (PP)	***Security Target*** (ST):
Conjunto de requisitos funcionales y de garantías independientes de implementación enfocadas a la identificación de un conjunto de objetivos de seguridad en un determinado dominio. Ejemplo: PP sobre un firewall, PP sobre un sistema de control de accesos, etc.	Conjunto de requisitos funcionales y de garantías que se usan como especificaciones de seguridad que debe satisfacer o proporcionar un producto o sistema. Ejemplo: ST para CheckPoint Firewall-1, para Oracle v.7, etc.

Cuadro 1
Fuente: Propia.

A continuación de definen los Requisitos Funcionales de Seguridad, los cuales están agrupados en las siguientes clases:

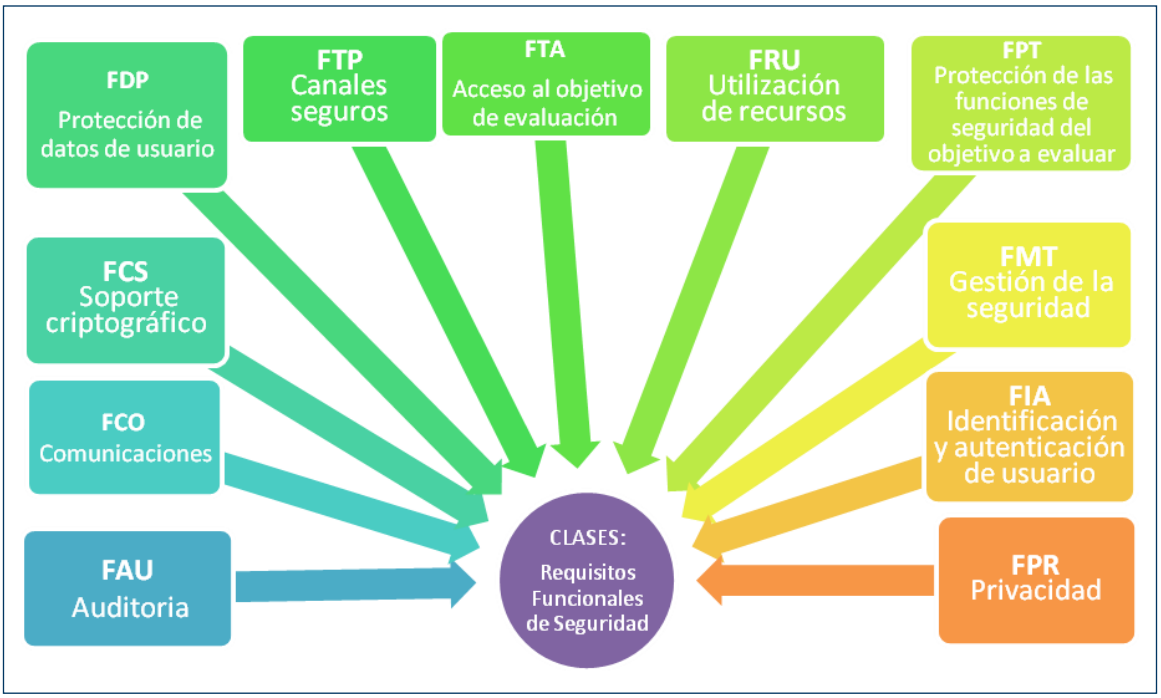


Imagen 1
Fuente: Propia.

Se continua con la definición de Requisitos de Garantía de Seguridad, los cuales definen los niveles de confianza que ofrecen las funciones de seguridad del producto o sistema. Estos se encuentran organizados bajo las siguientes clases:

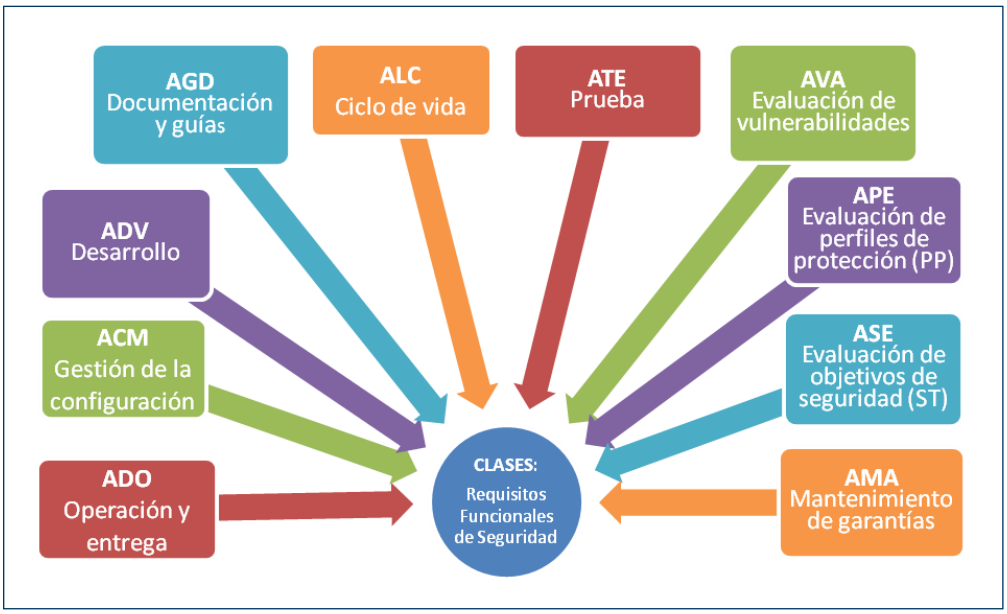


Imagen 2
Fuente: Propia.

Una vez se han definido los requisitos, clases, se debe iniciar un proceso de evaluación, el cual se representa en la siguiente imagen:

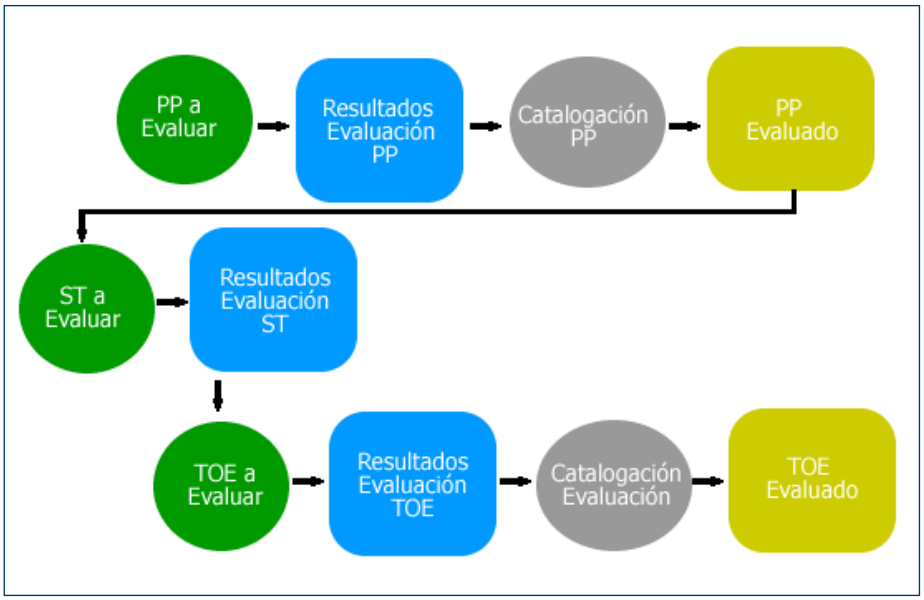


Imagen 3
Fuente: Propia.

Se inicia el proceso con una definición y documentación de los Objetivos de Evaluación (TOE) del producto o sistema a evaluar. En este documento también se relacionan los recursos y dispositivos que utiliza, la documentación que genera y el entorno en el que funcionara. Se pueden realizar los siguientes tipos de evaluación:

- Evaluación de Perfiles de Protección (PP): busca demostrar que un Protection Profile (PP) es completo, consistente y sólido. Además permite definir especificaciones de seguridad independientemente de la forma de implementación, por lo que pueden ser la base de especificaciones para otros productos o sistemas. Puede ser usado como punto de partida para definir requisitos destinados a establecer un Objetivo de Seguridad (ST).
- Evaluación de Objetivos de Evaluación (TOE): busca demostrar que todos los requisitos establecidos en el Security Target (ST) han sido implementados en el producto o sistema. Esto se hace tomando como base un Objetivo de Seguridad (ST) que ya ha sido evaluado.

Como resultado de la evaluación, se pueden certificar distintos Niveles de Seguridad (EAL). Gráficamente podemos ver estos niveles de la siguiente forma:

Niveles de seguridad (EAL)

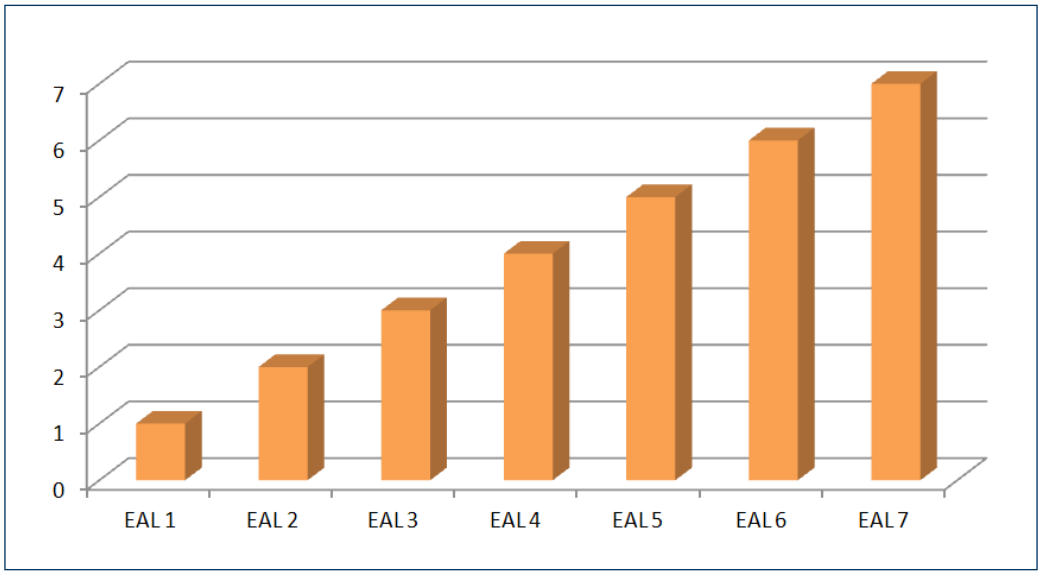


Imagen 4
Fuente: Propia.

EAL 1. Functionally tested

Nivel básico de seguridad, el cual se realiza analizando las funciones de seguridad y mediante el uso de especificaciones no formales res pecto a: Funcionalidades, de interfaz, guías y documentación del producto, con el fin de entender el comportamiento respecto a seguridad. Se aplica

cuando se necesita una confianza respecto a la correcta operación, pero las amenazas de seguridad no representan un gran peligro sobre el sistema.EAL 1 indica que las funciones de seguridad del TOE han sido implementadas consistentemente en la documentación y que existe una adecuada protección contra las amenazas que han sido identificadas

EAL 2. Structurally tested

Adicional a los requisitos del nivel EAL 1, requiere:

- Una descripción detallada e informal del diseño.
- Realización de pruebas en el desarrollo de acuerdo a las especificaciones funcionales y confirmarlas en forma independiente.
- Análisis de la fuerza de las funciones de seguridad implementadas y evidencias de la verificación de la respuesta del producto a las vulnerabilidades más comunes.
- Cooperación del equipo de desarrollo para la entrega de información sobre el diseño y resultados de pruebas.

EAL 2 es adecuado cuando desarrolladores o usuarios requieren cierto nivel de garantías de seguridad y no tienen acceso a toda la documentación generada en la fase de desarrollo.

EAL 3. Methodically tested and checked

Además de los requisitos de EAL 2, EAL3 obliga a:

- Que se implemente un desarrollo metódico determinado durante la fase de diseño.
- Usar controles de seguridad en los procesos de desarrollo para garantizar que el producto no ha sido manipulado durante su desarrollo. Para esto, se analizan las funciones de seguridad con base en las especificaciones funcionales de alto nivel, la documentación, guías del producto y los test obtenidos en la fase de prueba.

EAL 4. Methodically designed, tested and reviewed

Además de los requisitos de EAL 3, requiere de:

- Un análisis independiente de la vulnerabilidad, de tal forma que se pueda demostrar la resistencia contra intrusos que tengan un bajo potencial de ataque.
- Una especificación de bajo nivel del diseño de la implementación.

EAL 5. Semiformally designed and tested

- Requiere de descripciones semiformales del diseño y la arquitectura
- Una completa documentación de la implementación.
- Se debe realizar un análisis completo de vulnerabilidad, de tal forma que se pueda probar la resistencia frente a atacantes de potencial medio.
- Se deben mejorar los mecanismos de control, de tal forma que se garantice y se demuestre que durante el desarrollo, el producto no es manipulable frente a las especificaciones establecidas para el mismo.

EAL 6. Semiformally verified design and tested

Adicional a los requisitos del EAL 5, se requiere:

- Un análisis detallado de las funciones de seguridad.
- Una representación estructurada de su implementación.
- Una representación semiformal de la relación entre las especificaciones de alto y bajo nivel con la implementación.
- Demostrar que la robustez de las funciones de seguridad frente a atacantes de alto potencial de daño, han sido probadas durante el proceso de desarrollo.
- Que el desarrollo ha sido probado con un análisis de vulnerabilidades independiente.

EAL 7. Formally verified design and tested

Adicional a los requisitos de EAL 6, se debe:

- Probar formalmente las fases de desarrollo y prueba.
- Evaluar en forma independiente la confirmación de los resultados obtenidos de las pruebas realizadas para detectar vulnerabilidades durante el proceso de desarrollo.
- Demostrar la robustez de las funciones de evaluación.
- Realizarse un análisis independiente de vulnerabilidades para demostrar resistencia frente a un atacante de alto potencial.

Comprehesive Lightweight Application Security Process (CLASP)

Conceptos

- Es una actividad impulsada, mediante un conjunto basado en roles de componentes de procesos guiados por las mejores prácticas formalizadas.
- Está diseñado para ayudar a los equipos de desarrollo de software a construir la seguridad del producto, en las primeras etapas del ciclo de vida, de una manera estructurada, repetible y medible.
- Se basa en un gran trabajo de campo por parte del equipo de desarrollo, en el cual se descomponen los recursos del sistema de muchos ciclos de vida de desarrollo para crear un amplio conjunto de requisitos de seguridad. Los requisitos generados forman la base de CLASP's Best Practices (Mejores Prácticas de CLASP), las cuales permiten a las organizaciones a analizar y tratar sistemáticamente las vulnerabilidades de sus productos.

CLASP Views (Vistas CLASP)

Permiten a los usuarios CLASP entender de forma rápida el proceso CLASP, la interacción de sus componentes y como aplicarlos en el ciclo de vida del proceso de desarrollo.

Vista de Concepto. Proporciona una introducción de alto nivel a CLASP en cuanto a:

- Interacción de las cinco vistas.
- Siete mejores prácticas.
- Taxonomía.
- Relación con las políticas de seguridad.
- Ejemplos para aplicar componentes del proceso.

Vista Basada en Roles. Hace una introducción basadas en roles para el proceso CLASP.

Vista Actividad-Evaluación. Ayuda a los administradores de proyectos a evaluar la idoneidad de las 24 Actividades CLASP y seleccionar un subconjunto de ellos. CLASP ofrece dos medios para ayudar a seleccionar las actividades aplicables: La herencia y el nuevo inicio.

Vista Actividad-Implementación. Contiene las 24 Actividades CLASP relacionadas con la seguridad, las cuales se pueden integrar en un proceso de desarrollo de software. La fase de actividades del SDLC (System Development Life Cycle), se transforman en un ejecutable de software, de cualquier subconjunto de las 24 actividades relacionadas con la seguridad que fueron evaluadas y aceptadas en la Vista Actividad-Evaluación.

Vista de Vulnerabilidad. Contiene el listado de los 104 "tipos de problemas" identificados por CLASP que forman la base de las vulnerabilidades de seguridad en el código fuente de la aplicación. Un tipo de problema individual no siempre es una vulnerabilidad de seguridad; con frecuencia, es una combinación de problemas que crean una condición de seguridad que conduce a una vulnerabilidad en el código fuente.

Asociados a Vista de Vulnerabilidad se encuentran los Casos de Uso de la vulnerabilidad CLASP, los cuales representan condiciones en las que los servicios de seguridad son vulnerables a los ataques en la capa de aplicación. Proporcionan a los usuarios CLASP, ejemplos fáciles de entender, específicos de la relación entre la seguridad del código fuente y las posibles vulnerabilidades que pueden generarse en servicios de seguridad básicos.

A continuación se presenta la relación existente entre las diferentes vistas de CLASP.

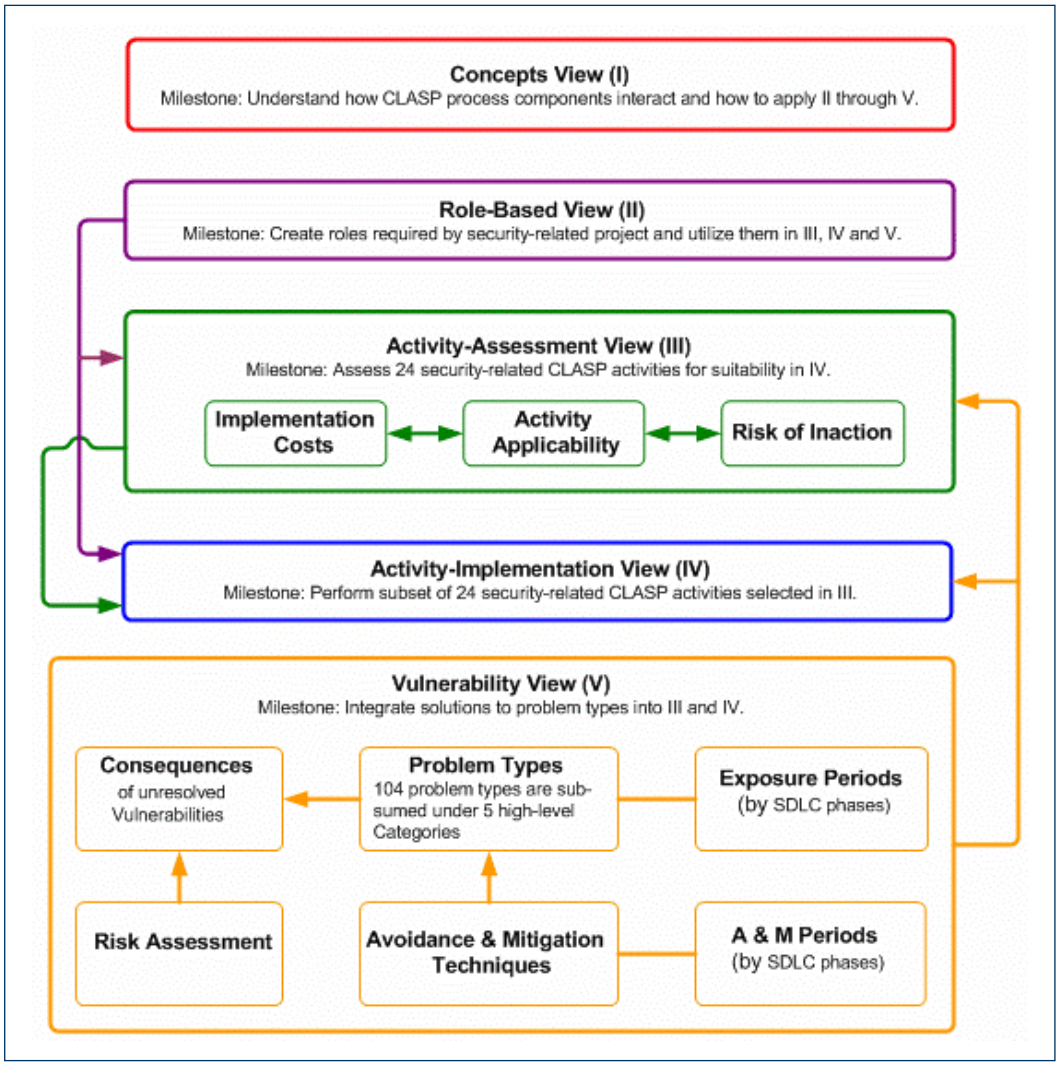


Imagen 4
Fuente: https://buildsecurityin.us-cert.gov/sites/default/files/clasp-figure1.gif

Las 7 Mejores prácticas de CLASP

Son la base de todos los programas de desarrollo relacionados con la seguridad de actividades, ya sea la planificación, el diseño o la implementación, incluyendo el uso de todas las herramientas y técnicas que apoyan CLASP. Estos son los siete CLASP Mejores Prácticas:

1. Instituir programas de sensibilización

   Es imprescindible educar al equipo de desarrolladores, gerentes y demás personas involucradas, en conceptos y técnicas de seguridad. Se pueden implementar programas de sensibilización, a través de la experiencia de expertos externos, según sea necesario y ayudando a asegurar que las actividades de promoción de software seguro se lleven a cabo de manera efectiva

2. Realizar evaluaciones de aplicación

   Las evaluaciones automatizadas pueden encontrar problemas de seguridad que no pueden ser detectados durante la codificación o pruebas de la aplicación; permiten encontrar riesgos de seguridad introducidos por el entorno operativo, y actuar como un mecanismo de defensa en profundidad por la captura de los fallos en el diseño, especificación, o implementación.

   Las funciones de prueba y evaluación son propias de un analista de prueba o por la organización de control de calidad, pero pueden abarcar todo el ciclo de vida.

3. Captura de requisitos de seguridad

   Asegurar que los requisitos de seguridad tienen el mismo nivel de "ciudadanía", como todos los demás "debe tener."

   Es fácil para los arquitectos de aplicaciones y administradores de proyectos centrarse en la funcionalidad a la hora de definir los requisitos, ya que soportan el mayor propósito de la aplicación para ofrecer valor a la organización. Las consideraciones de seguridad pueden ir fácilmente en el camino, por lo que es fundamental que los requisitos de seguridad sean una parte explícita de cualquier esfuerzo de desarrollo de aplicaciones. Entre los factores a considerar son:

   - Comprender como se usará la aplicación y cómo podría ser mal utilizada o bloqueada.
   - A cuales activos (datos y servicios) la aplicación dará acceso o que niveles de protección son apropiados, de acuerdo la exigencia de riesgo por parte de la organización, las regulaciones a las que está sujeta, y el impacto potencial sobre su reputación deben ser explotados en la aplicación.
   - La arquitectura de la aplicación y los tipos de ataque probables.
   - Posibles controles de compensación, su costo y efectividad.

4. Implementar prácticas de desarrollo seguras

   Para crear y mantener el código fuente reutilizable que permita fortalecer los servicios básicos de seguridad en una aplicación, se requiere de la implementación de prácticas de desarrollo de seguras en el proceso de desarrollo del SDLC. Las Mejores Practicas CLASP permiten:

- Poner a disposición un amplio conjunto de componentes de proceso.
- Proporcionar actividades basadas en roles bien definidos, permitiendo a los equipos de proyecto, tener una guía en la implementación de los principios de seguridad para diseñar, integrar el análisis de seguridad en el proceso de administración de origen, y la implementación / elaboración de las políticas de recursos y tecnologías de seguridad.
- Proporcionar muchos ejemplos de directrices de codificación, como los 104 Tipos de Problemas CLASP, que ayudan al equipo de proyecto detectar y resolver vulnerabilidades de seguridad en código fuente.

5. Construir procedimientos de enmendar vulnerabilidades

Acelera la respuesta y minimiza el riesgo mediante la definición de funciones, responsabilidades y procesos a seguir después de la identificación de la vulnerabilidad, en el contexto de actualizaciones y mejoras de las aplicaciones. Estos procesos permiten definir las medidas que se tomarán para identificar, evaluar, priorizar y remediar dichas vulnerabilidades y a menudo son alimentados por las evaluaciones de software, tanto en forma local o de un tercero, y ayudan a controlar la información que debe producirse para su divulgación.

6. Definir y monitorear las métricas

Los indicadores son un elemento esencial de un esfuerzo general de seguridad de la aplicación y son cruciales para evaluar la situación actual de seguridad de una organización. También ayudan a enfocarse en las vulnerabilidades más críticas, y revelan lo bueno y lo malo de la inversión de la organización en la mejora de la seguridad.

7. Publicar directrices de la seguridad operacional

La seguridad no termina cuando se ha completado una solicitud y desplegado en un entorno de producción. Para aprovechar al máximo la red existente y las inversiones en seguridad operacional se requiere que los encargados de la supervisión y la gestión de la seguridad del sistema, sean informados y educados; necesitan un asesoramiento y orientación sobre los requisitos de seguridad que requieren las aplicaciones y cómo optimizar las capacidades integradas en la aplicación.

Las 24 Actividades CLASP

Cada Actividad CLASP se divide en componentes de proceso discretos y vinculados a una o más funciones específicas del proyecto; de esta forma CLASP brinda una guía para que los participantes en el proyecto (jefes de proyecto, auditores de seguridad, desarrolladores, arquitectos, probadores, etc.) puedan adaptarlas fácilmente a su forma de trabajar. Esto genere mejoras incrementales a la seguridad que son fácilmente alcanzables, repetible y medible.

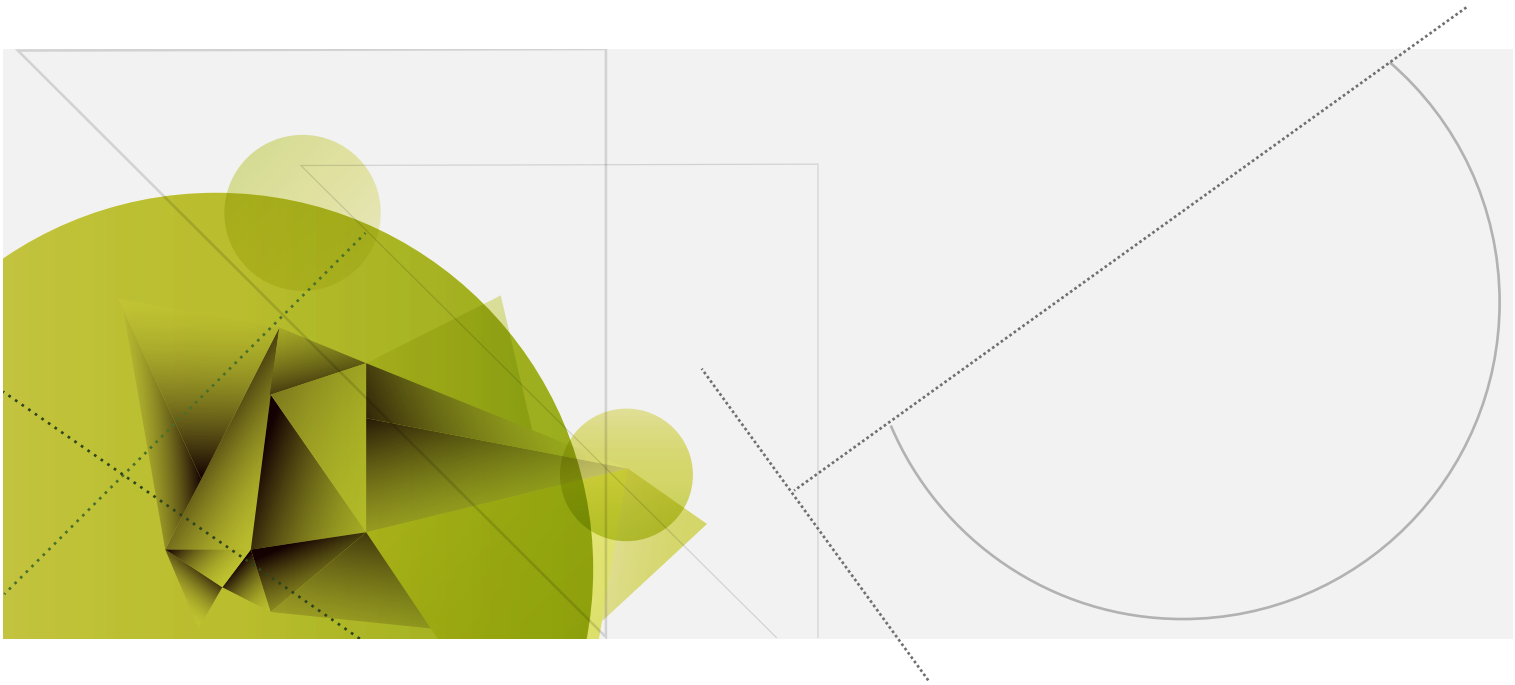
Mejores prácticas CLASP	Actividades CLASP	Roles de proyecto relacionados
1. Instituir programas de sensibilización	Instituir programas de concientización sobre la seguridad.	Gerente del proyecto.
2. Realizar evaluaciones de aplicación	Realizar análisis de la seguridad de los requisitos del sistema y el diseño (modelado de amenazas).	Auditor de seguridad.
	Realizar revisión de seguridad a nivel de fuente.	Propietario: auditor de seguridad. Colaborador clave: implementador, diseñador.
	Identificar, implementar y realizar pruebas de seguridad.	Analista de pruebas.
	Verifique los atributos de seguridad de los recursos.	Tester.
	Investigue y evaluar la postura de seguridad de soluciones tecnológicas.	Propietario: diseñador. Colaborador clave: proveedor de componentes.
3. Captura de requerimientos de seguridad	Identificar políticas de seguridad global.	Requisitos especificador.
	Identificar los recursos y los límites de confianza.	Propietario: Arquitecto. Colaborador clave: requisitos especificador.
	Identificar las funciones de usuario y capacidades de recursos.	Propietario: Arquitecto. Colaborador clave: requisitos especificador.
	Especificar entorno operativo.	Propietario: requisitos especificador. Colaborador clave: arquitecto.
	Detalle casos de uso indebido.	Propietario: requisitos especificadores. Factor clave: los grupos de interés.
	Identificar superficie de ataque	Diseñador.
	Requisitos de documentos de seguridad pertinentes.	Propietario: requisitos especificador. Colaborador clave: arquitecto.
4. Identificar practicas seguras de desarrollo	Aplicar principios de seguridad al diseño.	Diseñador.
	Anotar diseños clase con propiedades de seguridad.	Diseñador.
	Implementar y elaborar políticas de recursos y tecnologías de seguridad.	Implementador.
	Implementar contratos de interfaz.	Implementador.
	Integrar el análisis de seguridad en el proceso de gestión de la fuente.	Integrador.
	Realizar la firma de código.	Integrador.
Construir procedimientos de remediación de vulnerabilidades	Administrar proceso de divulgación problema de seguridad.	Propietario: director del proyecto. Colaborador clave: diseñado.
	Dirección reportó problemas de seguridad.	Propietario: diseñador reportero de falla.
Definir y monitorear las métricas	Monitorear métricas de seguridad.	Gerente de proyecto.
Publicar directrices de seguridad operativa	Especifique la configuración de seguridad de base de datos.	Diseñador de la base de datos
	Construir guía seguridad operativa.	Construir Guía dirección seguridad operativa.

Cuadro 2
Fuente: Propia.

Bibliografia

- Alcalde, E. & García, J. (1996). Introducción a la teleinformática. Colombia: McGraw-Hill.
- Jacobson, I. (2000). El proceso unificado de desarrollo de software. (3ª. Ed.). Londres: Pearson PLC.
- Long, L. (1996). Introducción a las computadoras y al procesamiento de información. (4ª. Ed.). México: Prentice Hall.
- Norton, P. (2000). Introducción a la computación. (3ª. Ed.). México: McGraw-Hill.
- Pressman, R. (2005). Ingeniería de software: un enfoque práctico. México: McGraw-Hill.

Esta obra se terminó de editar en el mes de noviembre
Tipografá Myriad Pro 12 puntos
Bogotá D.C,-Colombia.



AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO